Developer's Reference Manual

PC4500
PC4800

PC Card

Wireless LAN Adapter

Aironet™

Wireless Communications, Inc.

PC4500/PC4800

PC Card Wireless LAN Adapter

Developer's Reference Manual

No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Aironet Wireless Communications. Information in this document is subject to change without notice. Aironet Wireless Communications makes no representations or warranties with respect to the contents or use of this manual and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose.

© 1999, 1998, 1997 Aironet Wireless Communications. All rights reserved.

Aironet®, ARLAN® are registered trademarks of Aironet Wireless Communications, Inc. LM3000™, PC3000™, PC3500™, PC4500™ and PC4800™ are trademarks of Aironet Wireless Communications, Inc.

All names and product names used in this manual are trademarks or registered tradenames of their respective companies.

Printed in USA

DOC 710-004247, Revision B1

Contents

PART 1
INTRODUCTION
About the Developer's Reference Manual vii
Typographical Conventions ix
Reference Documents x
Welcome to the PC4500/4800 1-1
System Configurations 1-2
Coverage Options 1-2

PART 2
GETTING STARTED
Hardware Architecture 2-1
System Overview 2-1
Hardware Overview 2-2
Software Overview 2-3
Protocol Overview 2-3
Overview of the 802.11 Wireless LAN Protocol 3-1
Architecture Overview 3-1
802.11 Direct Sequence Frame Format 3-2
Basic Station Operation 3-5
Authentication and Key Management 3-7

PART 3
HARDWARE INTERFACING
PCMCIA Hardware Interface Operation 4-1
PCMCIA Hardware 4-1
PCMCIA Electrical Connections 4-2
ISA Hardware Interface Operation 5-1
ISA Hardware 5-1
ISA Mode 5-2
Serial Hardware Interface Operation 6-1
Serial Hardware 6-1
Serial Mode 6-1

PART 4
SOFTWARE INTERFACING
PCMCIA and ISA Client Software Interface 7-1
Host Memory Map 7-1
Host Attribute Memory 7-1
Host Common Memory 7-2
Host IO Registers 7-2
Recognizing the PC4500/4800 7-4
Bootstrap – Starting the PC4500/4800 7-5

Resetting the PC4500/4800 7-5
Interrupt Service Routine (ISR) processing 7-6
Enabling the PC4500/4800 7-6
Command and Status Register Descriptions 7-8
Command Register 7-12
Param0-2 Registers 7-12
Status Register 7-13
Resp0-2 Registers 7-13
Event Register Descriptions 7-13
EvStat Register 7-13
EvInt Register 7-15
EvAck Register 7-15
Basic FID Access 7-16
FID Management Register Descriptions 7-16
RxFID Register 7-16
TxAlloc FID Register 7-16
TxComplFID Register 7-17
Buffer Access Register Descriptions 7-17
Select0-1 Registers 7-17
Offset0-1 Registers 7-17
Data0-1 Registers 7-18
LinkStat Register 7-20
Host Software Register 7-21
SwSupport0-3 Registers 7-21
Host Auxiliary Registers 7-21
AuxPage Register 7-21
AuxOffset Register 7-21
AuxData Register 7-21
Command Descriptions 7-22
NOP Command 7-22
Enable Command 7-23
Disable Command 7-24
Lose Sync Command 7-24
Soft Reset Command 7-25
Host Sleep Command 7-25
Magic Packet Command 7-25
PSP Nodes Command 7-25
Allocate Transmit Buffer Command 7-25
Transmit Command 7-26
Deallocate Command 7-27
Access RID Command 7-27
Set PHY Register Command 7-28
Transmitter Test Command 7-29
Error Qualifier Values 7-31
Memory (FID/RID) Access 7-32
Reading and Writing RIDs 7-34

Frame Info Descriptor 7-35
FID with 802.3 Header – Station Mode 7-36
Receiving an 802.3 Packet 7-38
Transmitting an 802.3 Packet 7-39
FID without 802.3 Header – Access Point Mode 7-40
Receiving an 802.11 Packet 7-42
Transmitting an 802.11 Packet 7-43
FID Field Details 7-46
Resource Identifiers 7-49
General Configuration Parameters 7-50
Station Operation 7-53
Access Point Operation 7-54
Wireless LAN Monitor Operation 7-54
Packet Header Type 7-54
Payload Types 7-55
Aironet Extensions 7-55
Access Point Interface 7-55
Receive Mode 7-55
Device Type 7-55
802.11 Configuration Parameters 7-55
Power Save Operation 7-55
Modulation 7-56
WEP Key Volatile 7-56
WEP Key Non-Volatile 7-57
Valid SSID List 7-57
Valid AP List 7-58
Driver Name 7-58
Encapsulation Transformations 7-59
Capabilities RID 7-59
Status RID 7-61
Radio Information RID 7-61
Access Point RID 7-61
Statistics RID 7-62
Power Save Operation 7-66
PLAP Serial Client Software Interface 8-1
Introduction 8-1
Media Access Control 8-1
Autobaud Sequence 8-2
PLAP Synchronization 8-2
Receiving/Transmitting Frames 8-2
PLAP Frame Format 8-2
Migrating from the LM2000 to the PC4500/4800 8-4
PLAP Commands 8-4
SYNC_REQ 8-4
SYNC_ACK 8-5
DATA 8-5

CONFIGURE 8-6
DOWNLOAD 8-6
HOST_COMMAND 8-10
COMMAND_RESPONSE 8-12
PLAP Boot Strap of PC4500/4800 8-13

PART 5
REFERENCE
OEM Radio Approval Information 9-1
Safety Approvals 9-1
FCC Approvals 9-1
DOC Approvals (Canada) 9-2
ETSI Approvals (Europe) 9-2
OEM Labeling Requirements 9-4
US and US Territories 9-4
Canada 9-4
Europe 9-4
Other Countries 9-4
AWCSETEE Operation 9-5
Appendix A – PC4500/4800 Troubleshooting A-1
Indicator LEDs A-1
Troubleshooting A-1
Power Requirements A-2
Appendix B – PC4500/4800 PCMCIA CIS Description B-1
Appendix C – Reflashing the Firmware on the PC4500/4800 ..... C-1

List of Figures

Figure 1.1 - Minimal Overlap Coverage Option …… 1-3
Figure 1.2 - Heavy Overlap Coverage Option …… 1-3
Figure 1.3 - Multiple Overlapping Systems Coverage Option ……... 1-4
Figure 2.1 - Protocol Model ……... 2-2
Figure 2.2 - Hardware Block Diagram ……... 2-3
Figure 3.1 - Architecture Reference Model ……... 3-2
Figure 3.2 - PHY Header Description …… 3-3
Figure 3.3 - MAC Control Information ……. 3-3
Figure 3.4 - Relationship Between State Variables and Services ……... 3-5
Figure 8.1 - PLAP Packet Format …….. 8-3

List of Tables

Table A.1 - Green LED Operating Messages …… A-1
Table A.2 - Amber LED Operating Messages …….. A-1
Table A.3 - LED Error Codes ……... A-1
Table A.4 - Power Requirements …… A-2
Table B.1 - CIS Information ……. B-1

PREFACE

About the Developer's Reference Manual

This guide provides detailed information to aid a developer with both hardware and software interfacing to the PC4500 and PC4800.

Please consult the User's Guide & Technical Reference Manual, PC4500/4800 (DOC 710-004239) which covers installation and maintenance as well configuration with standard operating systems and network drivers before attempting to install or use the hardware and software described in this guide.

The Developer's Reference Manual is arranged as follows:

Part 1 - Introduction

Preface - About the Developer's Reference Manual - introduces the manual, conventions and reference materials.

Chapter 1 - Welcome to the PC4500/4800 - provides you with a general introduction to the PC4500/4800, direct sequence spread spectrum radio technology and various configurations you can use when employing the PC4500/4800 in your radio network.

Part 2 - Getting Started

Chapters in this section provide the information necessary to understand the PC4500/4800 basic theory of operation.

Chapter 2 - PC4500/4800 Architecture - provides you with an overview of PC4500/4800 architecture.

Chapter 3 - 802.11 Wireless LAN Protocol - provides you with an overview of the 802.11 Wireless LAN protocol and system tradeoffs.

Part 3 - Hardware Interfacing

Chapters in this section provide detailed hardware integration procedures for installing the PC4500/4800 using its various interfaces.

Chapter 4 - PCMCIA Interface - contains information necessary when designing a PCMCIA host interface to the PC4500/4800.

Chapter 5 - ISA Interface - contains information necessary when designing an ISA host interface to the PC4500/4800.

Chapter 6 - Serial Interface - contains information necessary when designing a serial host interface to the PC4500/4800.

Part 4 - Software Interfacing

Chapters in this section provide detailed software integration procedures for installing the PC4500/4800 using its various interfaces.

Chapter 7 - PCMCIA and ISA Client Software Interface - contains details for communicating to your PC4500/4800 using the PCMCIA and ISA interfaces.

Chapter 8 - PLAP Serial Client Software Interface - contains details for communicating to your PC4500/4800 using the serial interface via the PLAP protocol.

Part 5 –Reference

Appendix A - PC4500/4800 Troubleshooting - explains common problems encountered when integrating and configuring the PC4500/4800 for operation.

Appendix B - PC4500/4800 PCMCIA CIS Description - explains the PCMCIA configuration required by the PC4500/4800.

Appendix C - Upgrading Firmware on the PC4500/4800 - explains how to use the Aironet DOS utilities to upgrade firmware.

Typographical Conventions

When reading the Developer's Reference Manual, it is important to understand the symbol and formatting conventions used in the documentation. The following kinds of symbols and formatting are used in the guide.

Convention	**Type of Information**
	Indicates a note which contains important information set off from the normal text.
!	A caution message that appears before procedures which if not observed could result in loss of data or damage to the equipment.
Triangular bullet ➤	Indicates step-by-step procedures.
Bold type	An action you must perform such as type or select.
`Monospaced font`	Arguments and menus that are visible on the Configuration Software screens.
" "	Used to emphasize a word or term.

Reference Documents

Aironet Documents:

"Aironet Antenna Guide"
(document number 710-003725)

"User's Guide and Technical Reference Manual, PC4500/4800"
(document number 710-004239)

"User's Guide, AP4500/4800-E/T"
Ethernet and Token Ring versions (document number 710-004240)

"Technical Reference Manual, AP4500/4800-E/T"
Ethernet and Token Ring versions (document number 710-004242)

Third Party Documents:

IEEE Standard 802.11 "Wireless LAN Specification" by IEEE

"PC Card Standard" by the Personal Computer Memory Card Interface Association

IEEE Standard 996 "ISA Specification" by IEEE

"ISA and EISA Theory and Operation" by Edward Solari
ISBN 0-929392-15-9, third edition, 1994

1

INTRODUCTION

Welcome to the PC4500/4800

The PC4500 and the PC4800 are radio modules (typically used to create indoor wireless networks and limited distance outdoor applications) that provide transparent wireless data communications between a fixed, portable or mobile device and other wireless devices (such as access points connected to a wired network infrastructure). Host devices can be any device equipped with a PC Card Type II or Type III slot mechanical form factor. The host interface can be installed to operate as either a PC Card device, as a serial communications (UART) device, or as an ISA device.

When used in standard network applications, the PC4500/4800 is a direct replacement for a wired network interface card (NIC). The PC4500/4800 is fully Plug-and-Play compatible when used in host devices which support the technology.

The PC4500/4800 uses the 2.4GHz direct sequence spread spectrum technology as defined by the IEEE 802.11 Wireless LAN specification in order to accomplish real-time communication between a host computer or other device on a radio-based local area network. Direct sequence modulation provides secure, interference-immune communication and does not require a license for operation. The PC4500/4800 comes standard with Aironet enhancements to the 802.11 protocol which can be enabled to allow system optimizations which improve throughput and allow for more robust and reliable roaming.

The PC4500 is capable of transmitting and receiving at data rates of 1 and 2 Mbit/s, and the PC4800 can transmit and receive at data rates of 1, 2, 5.5, and 11 Mbit/s. The adapter can be configured to use either rate alone or automatically switch between both rates in order to maintain maximum throughput at a given range. Another advanced feature of the PC4500/4800 is the use of antenna diversity to offer improved data delivery (higher throughput) as well as extending the usable communication range of the PC4500/4800. Two antenna connectors are available which allows the connection of two antennas at the same time. These antennas can be configured as a single unit diversity antenna or two separate remote antennas. Connecting two antennas (or using the diversity antennas supplied by Aironet) allows the PC4500/4800 to detect and use the strongest signal coming from either of the antennas to improve receiver performance. The PC4500/4800 can be configured to use a single antenna for all transmissions or to choose the antenna offering the best data delivery. In this way, the PC4500/4800 provides you with the best communication range and reliability for your environment.

In addition to the secure and highly reliable data communication provided by direct sequence spread spectrum technology, the PC4500/4800 also offers a data encryption option. The encryption method is referred to as wired equivalent privacy (WEP) and is defined as an optional feature in the 802.11

standard. The PC4500/4800 WEP is capable of using 128 bit keys for additional security above the 802.11 standard which specifies the use of 40 bit keys.

System Configurations

The PC4500/4800 can be used in a variety of network system configurations. Aironet access points can be used to provide a wireless connection to an ethernet or token ring wired network. The PC4500/4800 can be used in any of the following 'standard' system architectures:

- Point to point wireless connectivity (computer to computer transfer)
- Point to multi-point configuration (cash register price updates from server)
- Multiple terminal communications (workgroup discussion)
- Multiple terminal communication using an access point root unit (conference room discussion)
- Wireless mobile infrastructure using access points off a wired LAN (warehouse and retail environments)
- Extended mobile infrastructure using repeaters (temporary parking lot sales)

An all wireless LAN is the simplest LAN configuration. These configurations are also referred to as peer to peer or ad-hoc networks (802.11 uses the term IBSS – independent basic service set). In an all Wireless LAN, using a peer to peer network operating system (such as Windows for Workgroups or Windows 95), all devices equipped with the PC4500/4800 can be linked together and communicate directly with each other.

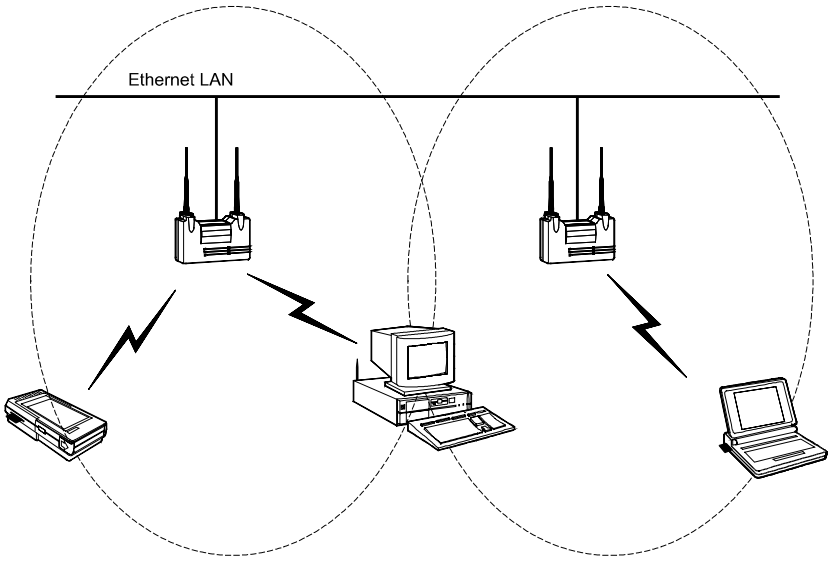
Better coordination can be achieved by adding an access point to the configuration. The access point is not required to be attached to any backbone LAN (such as an Ethernet LAN). It functions as a hub, linking all workstations and devices together. This configuration is very similar to the peer to peer network. The main difference is that the access point serves as the focal point for communications, thereby increasing the effective communication range, since all PC4500/4800 nodes are not required to be in direct communication range of each other. This configuration offers an added benefit of improved PC4500/4800 power management for increased battery life and operation of the host device.

A micro-cellular network (infrastructure) can be created by placing two or more access points on a backbone LAN. The wireless protocols and access point communications facilitate mobility by allowing remote workstations to move from the domain of one microcell to another. The process is seamless and transparent, and the connection to the file server or host is maintained without disruption. This configuration is particularly useful with portable or mobile network workstations allowing these nodes to maintain a session with an application executing on the wired network, even while moving about (roaming). When a network is configured using multiple access points and/or repeaters, a mobile client is automatically connected to the access point, which provides the best performance. This process is referred to a seamless roaming.

Coverage Options

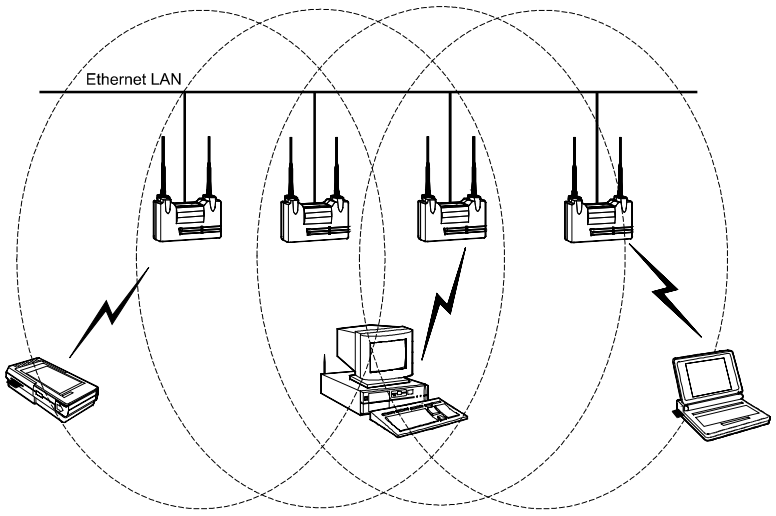
The system architecture options of the PC4500/4800 client devices and AP4500/4800 access points provide for a variety of coverage alternatives and flexibility. The system can be designed to provide a wide coverage area with minimal overlap (Figure 1.1) or coverage with heavy overlap (Figure 1.2) which can improve system performance and add protection (redundancy) against downtime in the event of an access point failure.

Figure 1.1 - Minimal Overlap Coverage Option



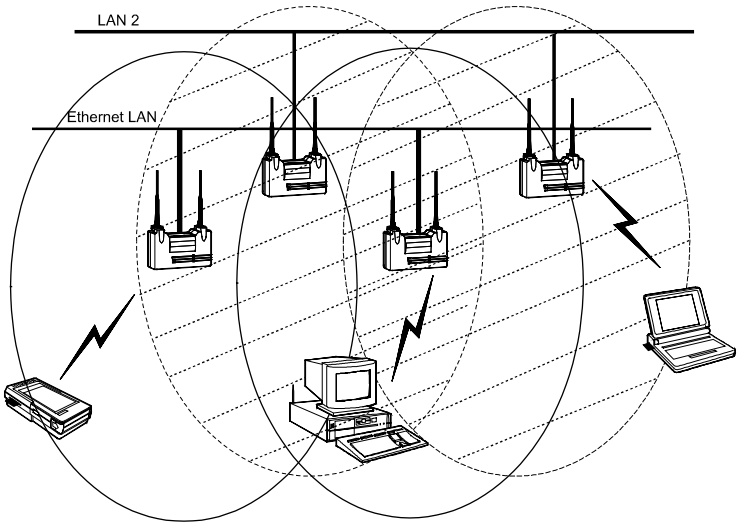
By arranging AP4500/4800 access points such that the overlap in coverage area is minimized, a large area can be covered with minimal system cost. The total bandwidth available to each mobile client will depend upon the amount of data each mobile client desires to transfer and the number of clients located in each cell as well as the data rates used for transmission. Seamless roaming is supported as a mobile client moves in and out of range of each access point, thereby maintaining a constant connection to the backbone network. Each access point (and PC4500/4800s) must be configured with the same system identifier in order to provide the roaming capability.

Figure 1.2 - Heavy Overlap Coverage Option



By arranging AP4500/4800 access points such that the overlap in coverage area is nearly maximized, a large number of mobile clients can be supported in the same general coverage area without any degradation in system performance or connect time. In addition, due to the redundancy in coverage overlap, system performance is not hampered in the event of an access point failure because clients will automatically 'roam' to an operational access point. With this architecture, all access points and PC4500\4800s must be configured with the same system identifier. The maximum available bandwidth will be the sum of that available from each AP4500/4800 with overlapping coverage areas.

Figure 1.3 - Multiple Overlapping Systems Coverage Option



Multiple systems can operate in the same vicinity by arranging AP4500/4800 access points such that the overlap in coverage area is as desired. The PC4500/4800-AP4500/4800 architecture provides multiple channels, which can exist in the same area with virtually no interference to each other. In this mode, each system must be configured with different system identifiers, which prevents PC4500/4800 clients from roaming to the access points of a different network. This architecture is useful in circumstances where different classes of users do not have access to the same network (such as manufacturing and inventory and design engineering may be on separate networks).

Refer to the User Guide and Technical Reference Manual, PC4500/4800 (DOC 710-004239) for additional details on these standard architectures and Aironet products.

PART 2

2

CHAPTER 2

PC4500/4800 Architecture

This chapter presents the high level design details of the PC4500/4800 and describes some of the features available from the hardware and software.

System Overview

The PC4500/4800 can be referred to as an RF modem and is responsible for handling the lowest two layers (Physical and Data Link) of the OSI protocol reference model. These layers are also referred to as the PHY (Physical) and MAC (Data Link) layers. The PHY layer is responsible for formatting the data into a form suitable for delivery as a radio signal. The MAC (media access control) layer is responsible for formatting the data and timing the delivery of the actual data packet.

Within the MAC layer, functions are divided into two levels. The lower level will handle the data delivery aspects of the wireless network including packet formatting and RF acknowledgment. The higher level is responsible for wireless network management. Some of the lower level functions are performed by hardware but most functionality is determined by software routines executing on a microcontroller protocol processor.

Executing on the host will typically be NDIS, ODI, or packet drivers which encompass the third and fourth layers of the OSI model (Network and Transport layers). These device drivers communicate to the protocol processor via an IO interface and/or interrupts.

PC4500/4800 Hardware Overview

The PC4500/4800 is a direct sequence spread-spectrum wireless network adapter. It uses a standard PC card interface, ISA interface, or a UART style serial interface to communicate to a host process. The main components of the PC4500/4800 architecture are: RF protocol processor, SRAM, Flash ROM, host interface, and direct sequence radio.

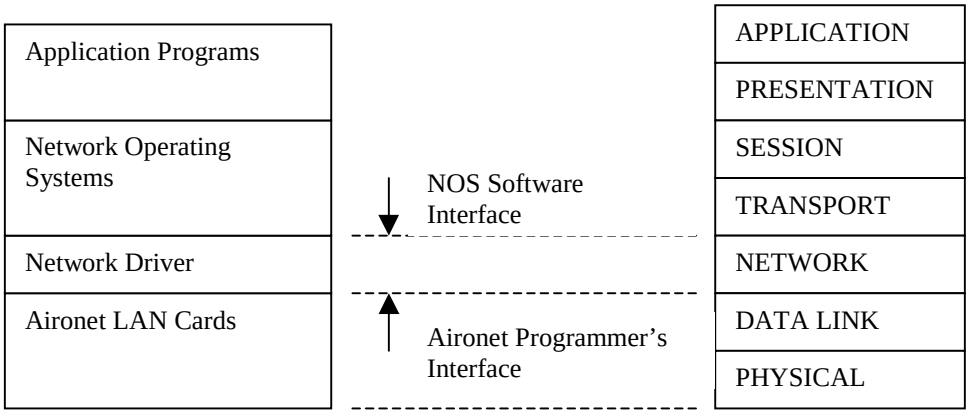


Figure 2.1 - Protocol Model

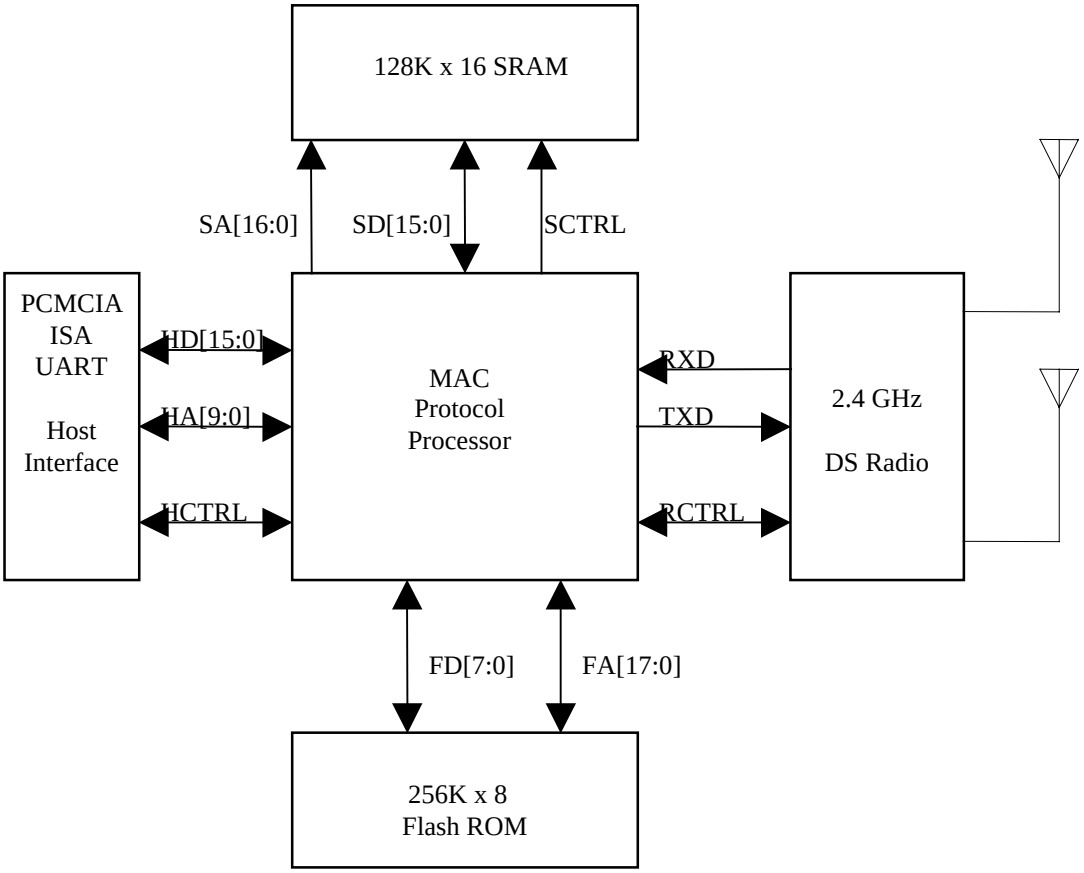


Figure 2.2 - Hardware Block Diagram

The RF protocol processor is responsible for the low-level protocol functions including the formation of the packet headers, data scrambling and descrambling, error checking, and packet acknowledgment as well as the accomplishment of the higher-level MAC functions and RF network management functions.

The 256KB (128Kx16) of SRAM is used for shared memory between the host processor and the PC4500/4800, buffer storage by the MAC processor, as well as stack, variable, and buffer space required by the MAC processor.

The 256KB of Flash memory can only be accessed by the MAC processor, and is used for program and radio calibration data storage.

The PC4500/4800 card has a serial prom containing a unique MAC ID for each card. This address can be overridden for applications requiring manually configured MAC addresses.

The PC4500/4800 supports three types of host interfaces: a standard PCMCIA interface, an ISA mode interface, or a UART style serial port.

The PC4500/4800 PCMCIA interface is compliant to the PCMCIA PC Card standard 3.0 electrical and physical specifications and dynamically responds to 16 bit and 8 bit accesses. This interface provides access to attribute memory (CIS configuration data) and I/O port (host registers). Access to the MAC controller registers and shared memory is via I/O addresses, which requires minimum resources from a host processor. Attribute memory is standardized to even-byte accesses. See Chapter 7 for a description of the host interface registers.

PC4500/4800 Software Overview

The PC4500/4800 implements a sophisticated wireless data protocol with a direct sequence spread-spectrum physical layer. This protocol is handled transparently to the user. The data delivery interface is accomplished through a command interface and transmit and receive data buffers.

PCMCIA and ISA operation requires a region of I/O space and an available interrupt. All data transfers to the PC4500/4800 use I/O reads and writes. Serial operation of the PC4500/4800 requires standard PC serial port drivers.

A section of memory is available which is arbitrated among the host process and the MAC processes running on the PC4500/4800. This memory contains the required command and status areas, transmit and receive buffers, and statistics and configuration information. These memory regions are mapped into the host's I/O space (usually via card and socket services) through 32 contiguous 16-bit I/O registers.

PC4500/4800 Protocol Overview

The PC4500/4800 implements a sophisticated wireless data protocol with a direct sequence spread-spectrum physical layer. For detailed information on the protocol, consult the IEEE 802.11 Wireless LAN specification. Chapter 3 contains an overview of the 802.11 protocol and highlights some extensions to the protocol provided by Aironet Wireless Communications.

The RF protocol used on the PC4500/4800 is a CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) system. This protocol provides a best effort packet delivery service with guaranteed data integrity by using extensive error checking (16-bit and 32-bit CRCs), length checks, sequence fields, packet flow control, and low-level packet acknowledgments. Once a directed packet is received by the PC4500/4800, an acknowledge is immediately sent informing the source that the packet was received correctly. If a packet is lost due to either a CRC error or a collision in the air with another packet, the sending station will not get an acknowledgment and the packet will automatically be retransmitted.

Upon successful reception of a complete packet, the PC4500/4800 will inform the host processor that it has a packet in its buffer. Duplicate packets are filtered out via the use of frame sequence numbers in the header of each frame.

The implemented RF protocol and media access techniques used by the PC4500/4800 cannot be altered by the programmer. Variations are limited primarily to setting the operating channel(s), maximum packet size, packet fragmentation size, flow control, and number of packet retries.

The PC4500/4800 can be set to operate in a direct peer-to-peer mode or as part of an infrastructure network. An infrastructure network consists of Aironet access points that are interconnected via a distribution system (which may be wireless or wired).

Beyond the point of determining if the PC4500/4800 is associated with an access point, the RF protocol is transparent to the driver residing on the host. The PC4500/4800 will automatically maintain an association and attempt to maintain the best link to the distribution system. The driver should monitor the association status of the PC4500/4800 to be sure that the association has not been lost.

3

CHAPTER 3

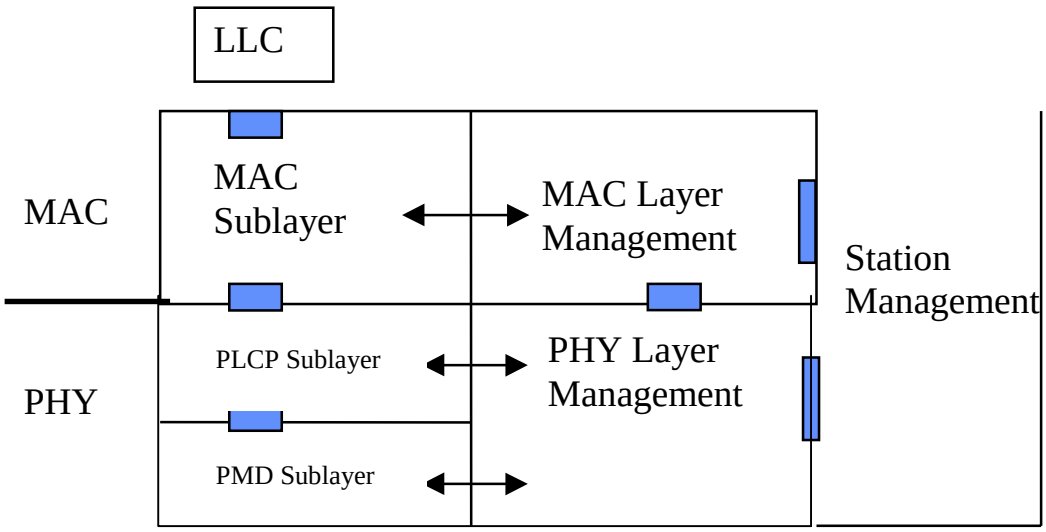
Overview of the 802.11 Wireless LAN Protocol

The IEEE 802.11 Wireless LAN Specification addresses only the lowest two layers of the protocol model. The main intent is to provide a framework such that multiple vendors can design interoperable products.

When compared to a wired protocol (like 802.3 for ethernet) which guarantees that all stations can directly communicate with each other, the uncertain nature of wireless communications does not guarantee that all users will be able to have direct communications with each other. To overcome this uncertainty, the 802.11 protocol is based on a MAC scheme known as CSMA/CA (carrier sense multiple access with collision avoidance). CSMA/CA allows for efficient use of the available bandwidth while arbitrating between all users.

Architecture Overview

Figure 3.1 - Architecture Reference Model



The MAC entity provides the basic access mechanisms to the RF medium and is responsible for functions such as encryption and fragmentation.

The MAC layer management is responsible for the synchronization functions, power management and roaming coordination.

The PHY layer management is primarily responsible for channel tuning and coordination of the radio functions.

The PLCP sublayer is responsible for clear channel assessment of the RF channel.

The PMD sublayer is responsible for the modulation and coding of the radio signals.

The station management entity interacts with both the PHY management and the MAC management to coordinate the desired overall station operation.

As identified in Chapter 1, the 802.11 specification supports three basic network topologies: the Independent Basic Service Set (IBSS), the Basic Service Set (BSS) and the Extended Service Set (ESS).

- The IBSS accomplishes peer to peer communication in which all stations desiring to communicate must establish a direct one to one connection.
- The BSS configuration uses an access point to provide communication coordination among multiple stations. The BSS typically has allows for a larger range than an IBSS as all parties are not required to be in direct communication with each other but are required to be in range of the access point.
- An ESS configuration is used to extend the wireless coverage area and consists of multiple BSS cells. Each cell can be linked via a wired backbone or the ESS can consist entirely of wireless communication paths.

IEEE 802.11 specifies the access point (AP) functionality required to coordinate the wireless station communication, however, it does not specify the functionality required to perform bridging functions (between a wired and the wireless network) or what to do with stored messages when a mobile client roams from one BSS to another. Aironet provides a consistent bridging procedure as well as message forwarding for power save and mobile stations.

There are many uncertainties which exist due to the nature of wireless communications such as interference and noise, hidden node problems and variations in the wireless link reliability. The 802.11 protocol provides several mechanisms to overcome these issues:

- Each frame is protected by a CRC to ensure data integrity.
- MAC layer acknowledge frames are used to indicate the successful delivery of a frame.
- RTS/CTS reservation can be used to ‘clear the way’ for a particular data exchange.
- Collision avoidance mechanisms are used to randomize subsequent delivery attempts of a failed frame.

IEEE 802.11 also provides a framework for power save operation which allows for a device to disable the radio functionality in order to conserve resources. Aironet provides several enhancements by which a user can establish the performance of power save operation by adjusting several competing parameters.

The 802.11 protocol also defines a mechanism through which the wireless transmissions can be encrypted as well as specifying an authentication procedure in order to protect against unauthorized network access. Aironet provides a higher level of encryption than what is specified by 802.11 and also provides end to end encryption functionality by offering a means for exchanging encryption keys.

802.11 Direct Sequence Frame Format

Figure 3.2 depicts the frame format used for each packet transmission.

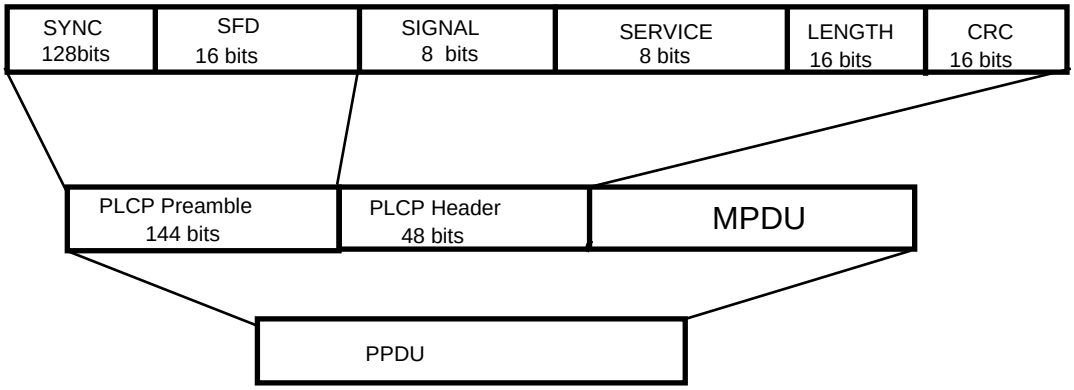


Figure 3.2 - PHY Header Description

Each frame consists of a PHY header which is always transmitted at the 1 Mbit/s data rate. The PHY header (which is protected by a CRC-16) indicates the data rate and length of the data portion of the payload. The PHY header is made up of the following components:

- A PLCP synchronization field which consists of 128 bits of scrambled ones. This portion of the signal is used to detect the presence of a signal, provide a waveform on which to perform antenna diversity as well as provide a constant pattern useful for determining accurate symbol timing.
- A Start Frame Delimiter (SFD) is a 16 bit pattern used to provide symbol level synchronization.
- An 8 bit 802.11 Signal Field indicates to the PHY the modulation which shall be used for transmission (and reception) of the MPDU. The data rate shall be equal to the Signal Field value multiplied by 100kbit/s. The DSSS PHY currently supports two mandatory modulation services given by the following 8 bit words, where the LSB shall be transmitted first in time:

    a) 0Ah for 1 Mbit/s DBPSK
    b) 14h for 2 Mbit/s DQPSK
    c) 37h for 5.5 Mbits/s CCK (PC4800 only)
    d) 6Eh for 11 Mbits/s CCK (PC4800 only)
- The 8 bit 802.11 service field shall be reserved for future use. The value of 00h signifies 802.11 device compliance.
- An unsigned 16 bit integer Length field used to indicate the number of microseconds (16 to 2^{16}-1) required to transmit the MPDU octets contained in the data portion of the frame.
- A Header Error Check field which is a 16 bit CRC applied over the length and signaling fields to ensure data integrity.

The PSDU portion of the frame contains the MAC data and control information. Each PSDU frame consists of the following basic components:

a) A *MAC Header*, which comprises frame control, duration, address and sequence control information.
b) A variable length *Frame Body*, which contains information specific to the frame type.
c) A *Frame Check Sequence (FCS)* which contains an IEEE 32-bit CRC.

The MAC Frame formats differ depending on the frame type (control, management or data). The basic format is shown in Figure 3.3. The fields Address 2, Address 3, Sequence Control, Address 4 and Frame Body are only present in certain frame types. Each field is defined in section 7.1.3 of the IEEE 802.11

specification. The format of each of the individual frame types is defined in section 7.2 of the IEEE 802.11 specification.

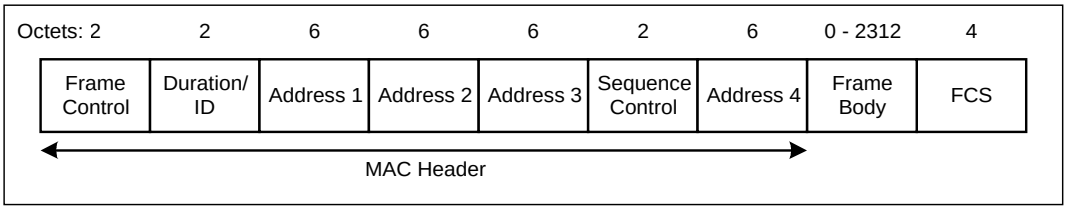


Figure 3.3 - MAC Control Information

The 2 byte Frame Control field consists of the following sub-fields: Protocol Version, Type, Subtype, To DS, From DS, More Fragments, Retry, Power Management, More Data, WEP and Order.

The Duration/ID field is 16 bits in length. The contents of the this field is as follows:

a) In Control Type frames of subtype PS-Poll, the Duration/ID field carries the association identity (AID) of the station that transmitted the frame in the 14 least-significant bits, with the 2 most-significant bits both set to 1. The value of the AID is in the range 1 - 2007.
b) In all other frames the Duration/ID field contains a duration value as defined for each frame type in section 7.2 of the 802.11 specification. For frames transmitted during the contention free period the duration field is set to 32768.

There are four address fields in the MAC frame format. These fields are used to indicate the BSSID, source address, destination address, transmitting station address and receiving station address. Each Address field contains a 48-bit address as defined in 5.2 of IEEE Std 802-1990.

The Sequence Control field is 16 bits in length and consists of two subfields, the Sequence Number and the Fragment number.

The Frame Body is a variable length field and contains information specific to individual frame types and subtypes. The minimum frame body is zero octets. The maximum length frame body is defined by the maximum length (MSDU + ICV + IV); where ICV and IV are the WEP fields defined in section 8.2.5 of the IEEE 802.11 specification.

The FCS field is a 32 bit field containing a 32-bit Cyclic Redundancy Check (CRC). The FCS is calculated over all the fields of the MAC header and the frame body field.

Table 3.1 indicates all of the defined 802.11 frame types.

Basic Station operation

A Station keeps two state variables for each station with which direct communication via the wireless medium is needed:

Authentication State:
The values are: Unauthenticated and Authenticated.
Association State:
The values are: Unassociated and Associated.

These two variables create three local states for each remote station:

State 1:
Initial start state, Unauthenticated, Unassociated.

State 2:
Authenticated, not Associated

State 3:
Authenticated and Associated.

The relationships between these station state variables and the station services are shown in Figure 3.4.

Table 3.1 – 802.11 Frame Types

Subtype Value b7 b6 b5 b4	Subtype Description
MANAGEMENT frames (type 00)	
0000	Association Request
0001	Association Response
0010	Reassociation Request
0011	Reassociation Response
0100	Probe Request
0101	Probe Response
0110-0111	Reserved
1000	Beacon
1001	ATIM
1010	Disassociation
1011	Authentication
1100	Deauthentication
1101-1111	Reserved
CONTROL frames (type 01)	
0000-1001	Reserved
1010	PS-Poll
1011	RTS
1100	CTS
1101	ACK
1110	CF-End
1111	CF-End + CF-Ack
DATA frames (type 10)	
0000	Data
0001	Data + CF-Ack
0010	Data + CF-Poll
0011	Data + CF-Ack + CF-Poll
0100	Null Function (no data)
0101	CF-Ack (no data)
0110	CF-Poll (no data)
0111	CF-Ack + CF-Poll (no data)
1000-1111	Reserved
Reserved (type 11)	
0000-1111	Reserved

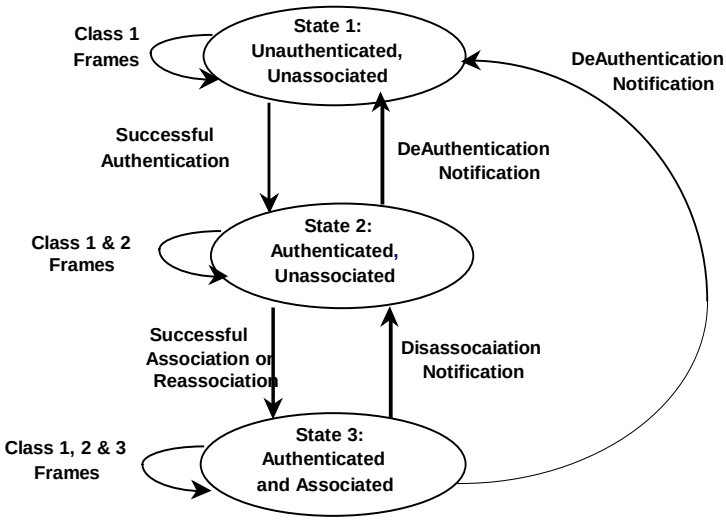


Figure 3.4 - Relationship Between State Variables and Services

The current state existing between the source and destination station determines the 802.11 frame types which may be exchanged between that pair of stations (see section 7 of the 802.11 specification). The state of the sending station given by Figure 3.4 is with respect to the intended receiving station. The allowed frame types are grouped into classes and the classes correspond to the station state. In State 1 only Class 1 frames are allowed. In State 2 either Class 1 or Class 2 frames are allowed. In State 3, all frames are allowed (Class 1, 2 and 3). The frame classes are defined as follows:

Class 1 frames (permitted from within States 1, 2 and 3):

Control Frames:

- RTS
- CTS
- ACK
- CF-End+ACK
- CF-End

Management Frames:

- Probe Request/Response
- Beacon
- Authentication

  Successful authentication enables a station to exchange Class 2 frames. Unsuccessful authentication leaves the Station in state 1.
- Deauthentication

  Deauthentication notification when in state 2 or state 3 changes the station's state to state 1.

  The station must become authenticated again prior to sending class 2 frames.
- ATIM

Data frames:

- Data

  Data frames with FC control bits "To DS" and "From DS" both false.

Class 2 frames (if and only if authenticated; allowed from within state 2 and state 3 only):

Management frames:

- Association Request/Response
  Successful association enables Class 3 frames.
  Unsuccessful association leaves station in state 2.
- Reassociation Request/Response
  Successful reassociation enables Class 3 frames.
  Unsuccessful reassociation leaves the station in State 2 (with respect to the station which was sent the reassociation message). Reassociation frames shall only be sent if the sending station is already associated in the same ESS.
- Disassociation
  Disassociation notification when in state 3 changes a station's state to state 2. This station must become associated again if it wishes to utilize the DS.

If station A receives a class 2 frame with a unicast address in the Address 1 field from station B which is not authenticated with station A, station A will send a deauthentication frame to station B.

Class 3 frames (if and only if associated; allowed only from within State 3):

Data frames:

- Data subtypes
  Data frames allowed. That is, either the "To DS" or "From DS" FC control bits may be set to true to utilize DS Services.

Management frames:

- Deauthentication
  Deauthentication notification when in state 3 implies disassociation as well, changing the station's state from 3 to 1. The station must become Authenticated again prior to another association.

Control frames:

- PS-Poll

If station A receives a class 3 frame with a unicast address in the Address 1 field from station B which is authenticated but not associated with station A, station A will send a disassociation frame to station B.

If station A receives a class 3 frame with a unicast address in the Address 1 field from station B which is not authenticated with station A, station A will send a deauthentication frame to station B.

(The use of the word "receive" refers to a frame which meets all of the filtering criteria specified in sections 8 and 9 of the 802.11 specification.)

PART 3

4

CHAPTER 4

PCMCIA Hardware Interface Operation

This chapter details the hardware aspects of the PC4500/4800 PCMCIA host interface. See Chapter 7 for details about the PCMCIA software interface.

PCMCIA Hardware

The PC4500/4800 PCMCIA interface is compliant to the PCMCIA PC Card standard electrical and physical specifications and fully supports 8 and 16 bit transfers. The tables below summarize the interface signals. For complete descriptions, see the PCMCIA PC Card standard. This interface provides access to attribute memory (CIS configuration data) and I/O memory (host registers).

Access to the MAC controller registers and shared memory is via 32 (16-bit) registers mapped into the host I/O space. The PC4500/4800 also requires a host interrupt for interrupt driven buffer processing. The provision of these resources is typically performed with the interaction of card and socket services and the host-side device driver.

For more complete information about the PCMCIA electrical interface, see the PCMCIA PC Card standard.

See Appendix C for descriptions about several utility programs provided on the OEM Developer disk which are useful for debugging the PC4500/4800 operation. These utilities allow the user to switch the PC4500/4800 operation between PCMCIA, ISA, and Serial modes, upgrade the PC4500/4800 firmware, and perform routine diagnostics.

PCMCIA Electrical Connections

PC4500/4800 Power Connections

PIN #	I/O	NAME	FUNCTION
1	DC	GND	Ground
34	DC	GND	Ground
35	DC	GND	Ground
68	DC	GND	Ground
17	DC	VCC	5V supply voltage
51	DC	VCC	5V supply voltage
18	DC	VPP1	Programming supply voltage, Host mode selection switch
52	DC	VPP2	Programming supply voltage, Host mode selection switch

Notes:

VPP – both VPP lines should be set to 5V on start up in order to indicate PCMCIA host mode.

PC4500/4800 Address Connections

PIN #	I/O	NAME	FUNCTION
11	I	A9	Address bit 9
12	I	A8	Address bit 8
22	I	A7	Address bit 7
23	I	A6	Address bit 6
24	I	A5	Address bit 5
25	I	A4	Address bit 4
26	I	A3	Address bit 3
27	I	A2	Address bit 2
28	I	A1	Address bit 1
29	I	A0	Address bit 0

Notes:

A[9:0] - Input signals for access to an attribute range of 256 even bytes (0-512) and an I/O space of 32 words as described in the software description. (A[9:6] are not used to access the I/O registers.)

PC4500/4800 Data Connections

PIN #	I/O	NAME	FUNCTION
41	I/O	D15	Data bit 15
40	I/O	D14	Data bit 14
39	I/O	D13	Data bit 13
38	I/O	D12	Data bit 12
37	I/O	D11	Data bit 11
66	I/O	D10	Data bit 10
65	I/O	D9	Data bit 9
64	I/O	D8	Data bit 8
6	I/O	D7	Data bit 7
5	I/O	D6	Data bit 6
4	I/O	D5	Data bit 5
3	I/O	D4	Data bit 4
2	I/O	D3	Data bit 3
32	I/O	D2	Data bit 2
31	I/O	D1	Data bit 1
30	I/O	D0	Data bit 0

Notes:

D[15:0] - Bidirectional data bus signals for even-byte access to the attribute memory as well as byte-only or word-only access to the host I/O space.

PC4500/4800 Control Connections

PIN #	I/O	NAME	FUNCTION
7	I	CE1#	Card Enable #1
42	I	CE2#	Card Enable #2
9	I	OE#	Output Enable
15	I	WE#	Write Enable
16	O	IREQ#	Interrupt Request (see notes)
33	O	IOIS16#	I/O port is 16-bit wide , pulled low on PC4500/4800
44	I	IORD#	I/O Read
45	I	IOWR#	I/O Write
58	I	RESET	Card Reset - pulled high on PC4500/4800
59	O	WAIT#	Extended Bus Cycle (see notes)
60	O	INPACK#	Input Port Acknowledge
61	I	REG#	Register Select and I/O Enable - pulled high on PC4500/4800
63	O	STSCHG#	Open collector output pulled high on PC4500/4800. Used in power saving mode to wake up the host.

Notes:

CE1#, CE2#, IORD#, IOWR#, OE#, REG#, WAIT#, WE# - Input signals for bus control as defined in the PCMCIA standard.

INPACK#, IOIS16#, STSCHG# - Output signals for bus control as defined in the PCMCIA standard.

RESET - Input signal for resetting the PC4500/4800. This signal must be asserted for a minimum of 20 microseconds.

IREQ# - Output signal that functions as the RDY/BSY line following the deassertion of the RESET signal until the PC4500/4800 is configured and ready for operation. All access to the PC4500/4800 must be held off until this signal goes high. Once the reset operation is complete (interrupts are enabled via the PCMCIA Configuration Options Register), this signal functions as a level-sensitive interrupt to the host.

The WAIT# signal must be fully supported. A fixed number of wait states will not be sufficient for the PC4500/4800 interface.

PC4500/4800 Card ID Connections

PIN#	I/O	NAME	FUNCTION
36	O	CD1#	Card Detect #1 - connected to ground on PC4500/4800
67	O	CD2#	Card Detect #2 – connected to ground on PC4500/4800

Notes:

The Card Detect pins are used by the socket controller to determine if a card is inserted into a PCMCIA slot.

PC4500/4800 Unused Connections

PIN #	I/O	NAME	FUNCTION
43	O	VS#1	
57	O	VS#2	
62	O	SPKR#	This pin is pulled high on the PC4500/4800
8	I	A10	
10	I	A11	
21	I	A12	
13	I	A13	
14	I	A14	
20	I	A15	
19	I	A16	
46	I	A17	
47	I	A18	
48	I	A19	
49	I	A20	This pin is used as DTR for Serial mode operation
50	I	A21	This pin is used as RTS for Serial mode operation
53	I	A22	This pin is used as CTS for Serial mode operation
54	I	A23	This pin is used as DCD for Serial mode operation
55	I	A24	This pin is used as RXD for Serial mode operation
56	I	A25	This pin is used as TXD for Serial mode operation

Notes:

The above list of unused pins are ignored (or not used) during normal PCMCIA mode operation.

5

CHAPTER 5

ISA Hardware Interface Operation

This chapter details the hardware aspects of the PC4500/4800 ISA host interface. See Chapter 7 for details about the ISA software interface.

ISA Hardware

The PC4500/4800 ISA interface functions as either an 8-bit (byte only) or a 16-bit (word only) I/O interface device. The interface does not adhere to all signal definitions according to the ISA specification as some signals must be driven according to the PCMCIA specification. External address decode circuitry and control signal gating is required. The tables below summarize the interface signals. This interface provides access to the PC4500/4800 I/O memory (host registers).

Access to the MAC controller registers and shared memory is via 32 (16-bit) registers mapped into the host I/O space. The PC4500/4800 also requires a host interrupt channel for communication synchronization. The provision of these resources are typically performed by the host hardware and the host-side device driver.

For more complete information about the ISA electrical interface, refer to IEEE specification 996 or the text by Edward Solari (see the Applicable Documents section on page x).

ISA Mode

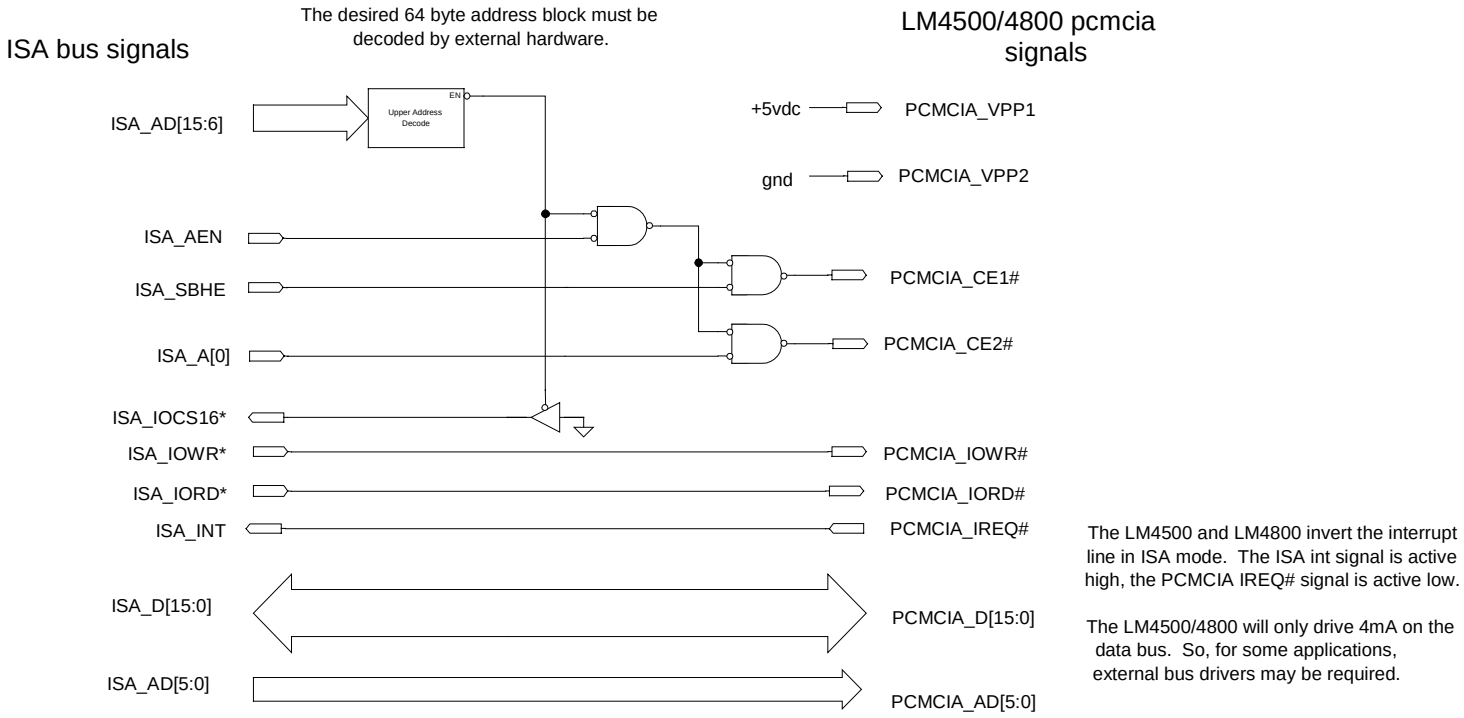
ISA mode on the PC4500/4800 uses PCMCIA timing and signals. It is called ISA mode in the sense that the PCMCIA configuration registers are not accessed in order to enable the card. The PC4500/4800 supports ISA mode by tying the VPP2 PCMCIA pin low at reset in order to enable the host interface. In this mode, the PCMCIA configuration registers are disabled and cannot be accessed. Likewise, the

Configuration Options register (COR) does not need to be written to access the card. All other interface registers act the same. All signals and timings are the same as in PCMCIA mode except that the sense of the interrupt line becomes active high in ISA mode.

In ISA mode, the PC4500/4800 requires external hardware to determine the IRQ and I/O base address that the card uses. The main requirements are the decoding of the upper address lines to generate a chip select and route the interrupt line to the desired interrupt. IOIS16# is always decoded as the cards can always respond to 16-bit accesses. The difference for ISA mode, is that the ISA bus requires this signal to be wire or'd with the signals from other cards on the ISA bus. This requires that an open collector gate be used.

The PC4500/4800 has an onboard reset controller which insures that the radio comes up properly after +5V is applied to the card. However, this does not preclude the use of an external reset circuit.

The following diagram illustrates the use of the PC4500/4800 in ISA mode.



The LM4500 and LM4800 invert the interrupt line in ISA mode. The ISA int signal is active high, the PCMCIA IREQ# signal is active low.

The LM4500/4800 will only drive 4mA on the data bus. So, for some applications, external bus drivers may be required.

PIN	PCMCIA definition	ISA definition	Comments
1	GND	GND	
2	D3	D3	
3	D4	D4	
4	D5	D5	
5	D6	D6	
6	D7	D7	
7	CE1#	CE1#	Lower byte of CS (active low) ; CE1 and CE2 are both driven low for a 16 bit access
8	A10	N/C	
9	OE#	MEMR	
10	A11	N/C	
11	A9	A9	
12	A8	A8	
13	A13	N/C	
14	A14	N/C	
15	WE#	MEMW	(See Note 4)
16	IREQ#	INT0	
17	VCC	VCC	
18	VPP1	VCC	
19	A16	A16	
20	A15	A15	
21	A12	A12	
22	A7	A7	
23	A6	A6	
24	A5	A5	
25	A4	A4	
26	A3	A3	
27	A2	A2	
28	A1	A1	
29	A0	A0	
30	D0	D0	
31	D1	D1	
32	D2	D2	
33	IOIS16#	IOCS16#	Tied low on the PC4500/4800 (See Note 2)
34	GND	GND	
35	GND	GND	
36	CD1#	N/C	
37	D11	D11	
38	D12	D12	
39	D13	D13	
40	D14	D14	
41	D15	D15	
42	CE2#	CE2#	Upper byte of CS (active low); CE1 and CE2 are both driven low for a 16 bit access
43	VS1	N/C	
44	IORD#	IOR#	I/O Read Strobe, active low
45	IOWR#	IOW#	I/O Write Strobe, active low , pulled high (See Note 4)
46	A17	N/C	
47	A18	N/C	
48	A19	N/C	
49	A20	N/C	
50	A21	N/C	

51	VCC	VCC	
52	VPP2	GND	Sets ISA mode; enabling interface
53	A22	N/C	
54	A23	N/C	
55	A24	N/C	
56	A25	N/C	
57	VS2	N/C	
58	RESET	RESET	Active high
59	WAIT#	IOCHRDY	Active low (See Note 3)
60	INPACK	N/C	(See Note 5)
61	REG#	REG#	Must be driven low according to PCMCIA spec
62	SPKR	N/C	
63	STSCHG	STSCHG	Open collector output signal
64	D8	D8	
65	D9	D9	
66	D10	D10	
67	CD2	N/C	
68	GND	GND	

Notes:

[1] N/C indicates not connected.

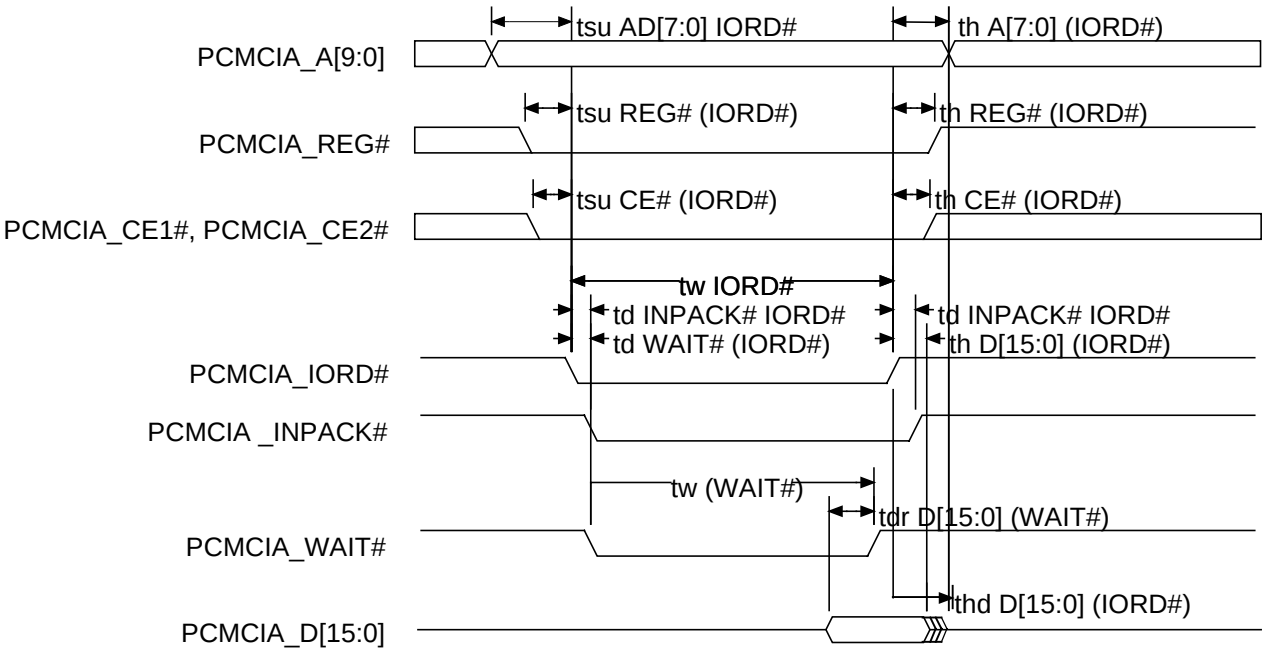
[2] The PC4500/4800 fully supports 8 bit I/O as well as 16-bit I/O. Since IOIS16# is tied to ground, it is not shown on the timing diagrams.

[3] Support of the WAIT# signal is required. Typical pulse widths for this signal are 250 to 500 nsec, but can be as long as 875 nsec. Accesses to hardware base registers will have a WAIT# pulse width of approximately 125nsec.

[4] Both memory writes and I/O writes require a minimum valid data setup time of 15nsec ahead of the rising edge of MEMW# or IOWR#.

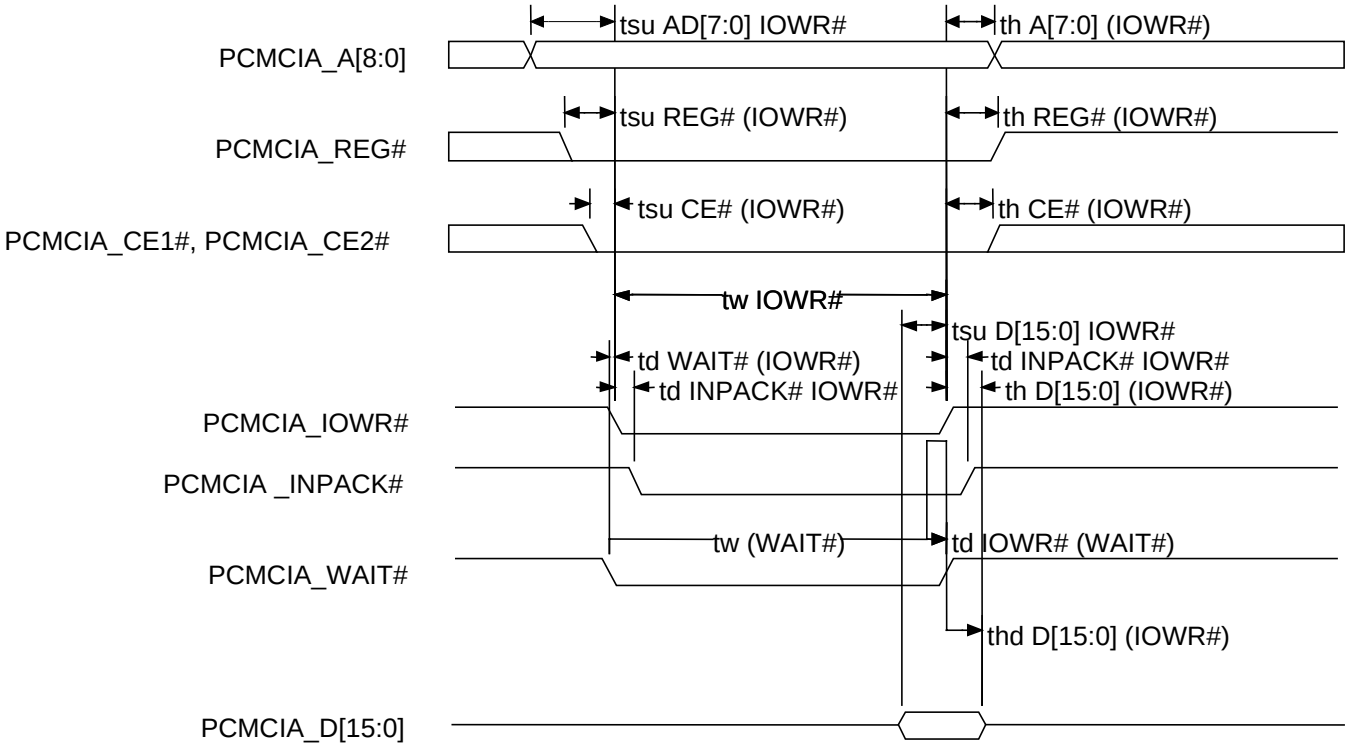
[5] The INPACK signal is returned for all I/O accesses. The use of this signal is not required since the address is fully decoded by the socket controller.

I/O Read Cycle



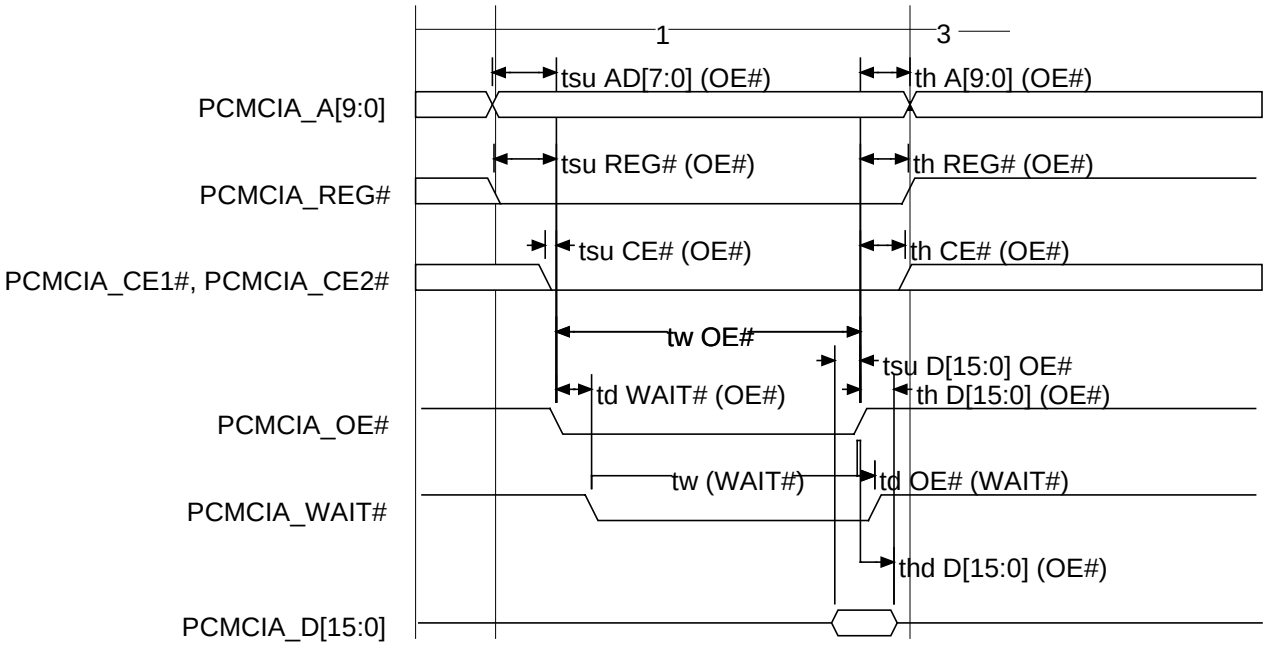
NAME	Min (ns)	Max (ns)	Comment
tw (WAIT#)	125	12,000	WAIT# pulse width
tw (IORD#)	165		IORD# pulse width
tsu REG# (IORD#)	5		REG# setup before falling IORD#
tsu CE# (IORD#)	30		CE# setup before falling IORD#
tsu A[7:0] (IORD#)	30		A[7:0] setup before falling IORD#
td INPACK# (IORD#)		45	INPACK# delay before falling IORD#
td WAIT# (IORD#)		35	WAIT# delay following IORD#
th A[7:0] (IORD#)	20		A[7:0] hold following rising IORD#
th REG# (IORD#)	0		REG# hold following rising IORD#
th CE# (IORD#)	20		CE# hold following rising IORD#
th D[15:0] (IORD#)	0		D[15:0] hold following rising IORD#
tdr D[15:0] (WAIT#)	0		D[15:0] delay following rising WAIT#
thd D[15:0] (IORD#)	30		D[15:0] hold following rising IORD#

I/O Write Timing



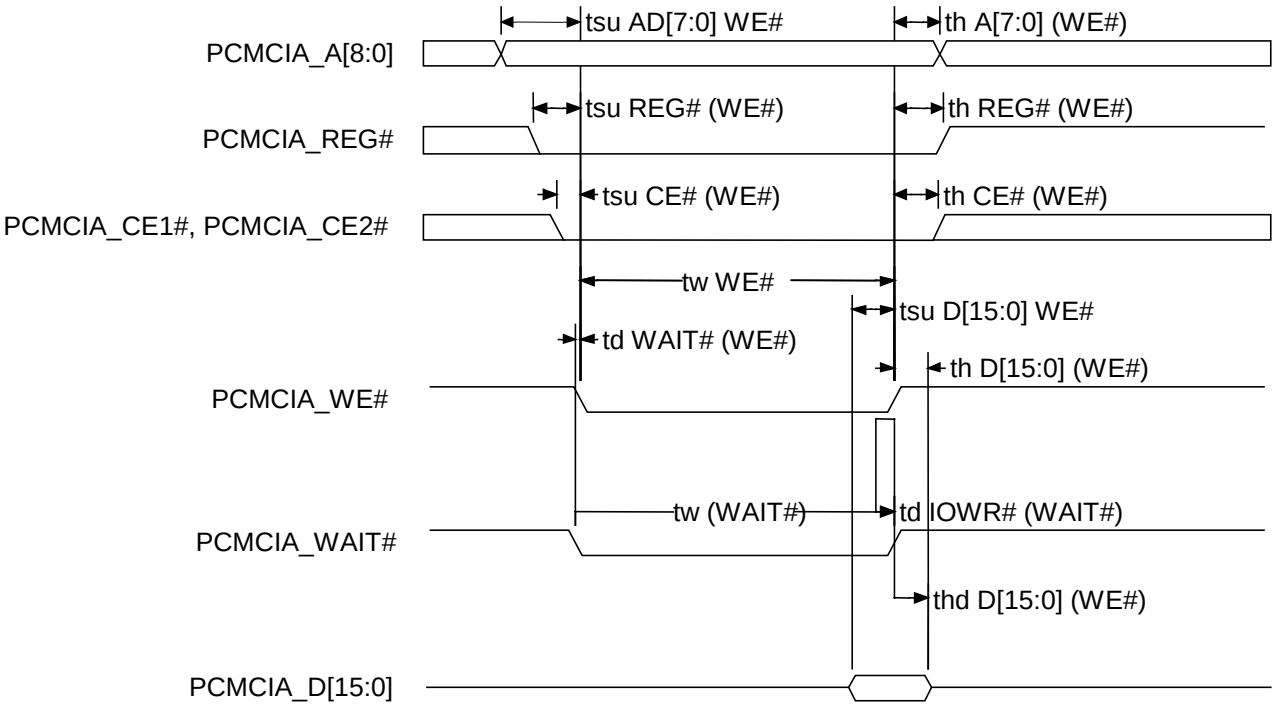
NAME	Min (ns)	Max (ns)	Comment
tw (WAIT#)	125	12,000	WAIT# pulse width
tw (IOWR#)	165		IOWR# pulse width
tsu REG# (IOWR#)	5		REG# setup before falling IOWR#
tsu CE# (IOWR#)	30		CE# setup before falling IOWR#
tsu A[7:0] (IOWR#)	30		A[7:0] setup before falling IOWR#
td INPACK# (IOWR#)		45	INPACK# delay before falling IOWR#
td WAIT# (IOWR#)		35	WAIT# delay following IOWR#
th A[7:0] (IOWR#)	20		A[7:0] hold following rising IOWR#
th REG# (IOWR#)	0		REG# hold following rising IOWR#
th CE# (IOWR#)	20		CE# hold following rising IOWR#
th D[15:0] (IOWR#)	0		D[15:0] hold following rising IOWR#
thd D[15:0] (IOWR#)	30		D[15:0] hold following rising IOWR#
td IOWR# (WAIT#)	0		IOWR# delay from rising WAIT#
tsu D[15:0] (IOWR#)	15		D[15:0] setup before falling IOWR#

Attribute Memory Read Timing



NAME	Min (ns)	Max (ns)	Comment
tw (WAIT#)	250	12,000	WAIT# pulse width
tw (OE#)	150		OE# pulse width
tsu REG# (OE#)	5		REG# setup before falling OE#
tsu CE# (OE#)	0		CE# setup before falling OE#
tsu A[7:0] (OE#)	30		A[7:0] setup before falling OE#
td WAIT# (OE#)		35	WAIT# delay following OE#
th A[9:0] (OE#)	20		A[7:0] hold following rising OE#
th REG# (OE#)	0		REG# hold following rising OE#
th CE# (OE#)	20		CE# hold following rising OE#
th D[15:0] (OE#)	0		D[15:0] hold following rising OE#
thd D[15:0] (OE#)	30		D[15:0] hold following rising OE#
td OE# (WAIT#)	0		OE# delay from rising WAIT#
tsu D[15:0] (OE#)	15		D[15:0] setup before falling OE#

Attribute Memory Write Timing



NAME	Min (ns)	Max (ns)	Comment
tw (WAIT#)	125	12,000	WAIT# pulse width
tw (WE#)	165		WE# pulse width
tsu REG# (WE#)	5		REG# setup before falling WE#
tsu CE# (WE#)	30		CE# setup before falling WE#
tsu A[7:0] (WE#)	30		A[7:0] setup before falling WE#
td WAIT# (WE#)		35	WAIT# delay following WE#
th A[7:0] (WE#)	20		A[7:0] hold following rising WE#
th REG# (WE#)	0		REG# hold following rising WE#
th CE# (WE#)	20		CE# hold following rising WE#
th D[15:0] (WE#)	0		D[15:0] hold following rising WE#
thd D[15:0] (WE#)	30		D[15:0] hold following rising WE#
td IOWR# (WAIT#)	0		IOWR# delay from rising WAIT#
tsu D[15:0] (WE#)	15		D[15:0] setup before falling WE#

6

CHAPTER 6

Serial Hardware Interface Operation

This chapter details the hardware aspects of the PC4500/4800 Serial host interface. See chapter 8 for details about the available software interfaces.

Serial Hardware

The PC4500/4800 PC card offers a Serial Interface mode to the host device by re-mapping the upper PCMCIA address lines to a UART-type serial interface. This allows the PC4500/4800 to be embedded and interfaced to via a serial port. The serial interface can be accessed using a serial protocol referred to as PLAP.

Serial Mode

The Serial mode is invoked automatically by the PC4500/4800 PC card upon a read of the VPP lines as listed in Table 6-2, and the reception of the autobaud sequence.

A power-on reset signal is provided internally. No external reset signal is required. The standard PCMCIA Card Reset signal (pin 58) is still available, being wire OR'd with the internal reset. This arrangement is present in both PCMCIA and Serial modes. Typical Serial mode applications require pin 58 to be driven low.

Table 6-1 lists the operating parameters of the PC4500/4800 Serial Mode interface.

Serial Port Mode Specifications

ITEM	SPECIFICATION
Baud Rates	7200 bit/s to 115.2 kbit/s using no parity, 8 data bits, 2 stop bits
Voltage Levels	TTL levels supported ViL < 0.8V VoL < 0.3V ViH > 2.0V VoH > 3.0V
Signal Polarity	RXD, TXD (NRZ data, non inverted) CTS, RTS (active low) DSR, RING (active low) BUSY_IN, BUSY_OUT (active high)

When Serial mode is required, the PC Card connections should be as shown in Table 6-2.

Serial Mode Pinouts

PIN#	PCMCIA NAME	SERIAL MODE CONNECTION	FUNCTION
1	GND	Grounded	
2	D3	Grounded	
3	D4	Grounded	
4	D5	Grounded	
5	D6	Grounded	
6	D7	Grounded	
7	CE1#	Tied high	
8	A10	N/C	
9	OE#	Tied high	
10	A11	N/C	
11	A9	Grounded	
12	A8	+5V	
13	A13	N/C	
14	A14	N/C	
15	WE#	Tied high	Tie to VCC (no pull-up required)
16	IREQ#	N/C	
17	VCC	apply VCC	
18	VPP1	+5V	Used as Mode pin
19	A16	N/C	
20	A15	N/C	
21	A12	N/C	
22	A7	Grounded	
23	A6	Grounded	
24	A5	Grounded	
25	A4	Grounded	
26	A3	+5V	
27	A2	Grounded	
28	A1	Grounded	
29	A0	Grounded	
30	D0	Grounded	
31	D1	Grounded	
32	D2	Grounded	
33	IOIS16#	N/C	
34	GND	Grounded	

35	GND	Grounded	
36	CD#1	N/C	Connected to ground on the PC4500/4800
37	D11	Grounded	
38	D12	Grounded	
39	D13	Grounded	
40	D14	Grounded	
41	D15	Grounded	
42	CE2#	Tied high	
43	VS1#	N/C	
44	IORD#	Tied high	Tie to VCC (no pull-up required)
45	IOWR#	Tied high	
46	A17	N/C	
47	A18	RING	RING is output from card (active low)
48	A19	DSR	DSR is output from card (active low)
49	A20	DTR/BO	DTR/BUSY OUT is input to card (active high)
50	A21	RTS#	RTS is input to card (active low)
51	VCC	Apply VCC	
52	VPP2	Grounded	Mode input
53	A22	CTS#	CTS is output from card (active low)
54	A23	DCD/BI	DCD/BUSY IN is output from card (active high)
55	A24	RXD	RXD is input to card (NRZ data)
56	A25	TXD	TXD is output from card (NRZ data)
57	VS2	N/C	
58	RESET	RESET	Optional input – OR'd with internal reset controller signal (active high); use minimum 20 usec pulse, pulled high on PC4500/4800. See note below.
59	WAIT#	N/C	
60	INPACK#	N/C	
61	REG#	Tied high	
62	SPKR#	N/C	
63	STSCHG#	O.C. output	Open collector output – Can be used for wake up event
64	D8	Grounded	
65	D9	Grounded	
66	D10	Grounded	
67	CD2#	N/C	Connected to ground on the PC4500/4800
68	GND	Grounded	

Note: the RESET signal is pulled high on the PC4500/4800, therefore, for the card to boot in serial mode, this signal must be driven low.

The following page provides a schematic to be used for interfacing the PC4500/4800 using a serial interface. This is a general drawing intended to be used with several Aironet PCMCIA card devices. The address and data lines of the PCMCIA connector can be left unconnected or may be tied to ground for proper PC4500/4800 operation.

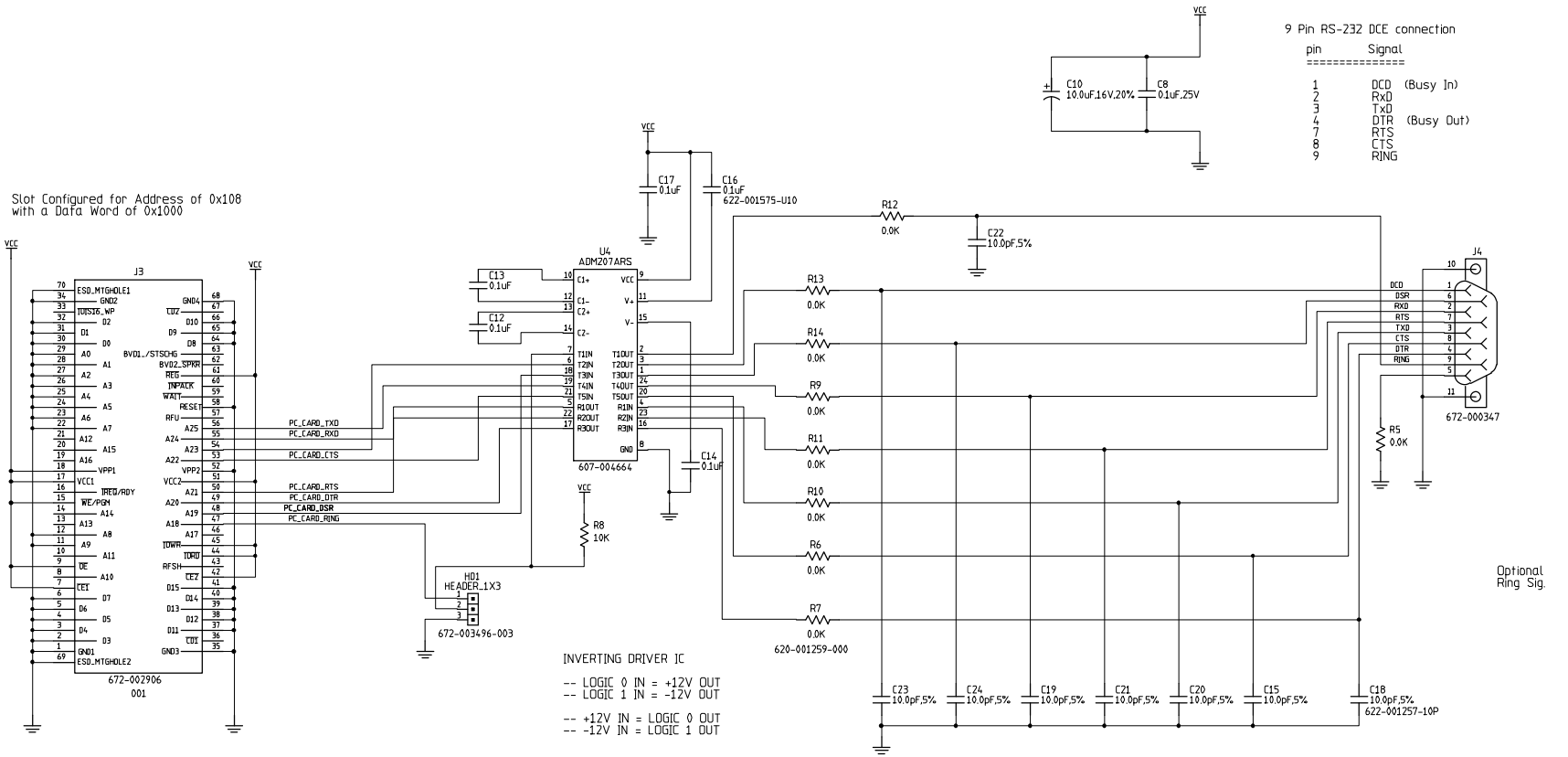
The following table gives some part number cross references.

Aironet Part #	Description	Vendor	Vendor #
670-002859	Right Angle Switch	MORS Components	MJTP1236E
672-000347	DB9 female connector	AMP Incorporated	787844-4
672-002906	68 pin PCMCIA socket	Berg Electronics	92789-050
672-003601	4 pin power connector	Weidmuller Inc.	151286

Serial Port Schematic

A suggested schematic for a serial port is shown on the following page.

REVISIONS				
REV	ECN NO.	DESCRIPTION	DATE	APPROVED



CONFIDENTIAL

DRAWING NUMBER	SIZE	SHEET
	B	

PART 4

7

CHAPTER 7

PCMCIA and ISA Client Software Interface

This chapter details the software interface to the PC4500/4800. Refer to chapters 4 and 5 for details about the PCMCIA and ISA hardware interfaces.

The main software components in a PCMCIA environment are Card Services and Socket Services. The use of these services provides sufficient abstraction to allow the use of multiple PC cards through different PCMCIA interface hardware. Socket Services provides actual PCMCIA hardware interface support and Card Services provides client processes access to the PC cards that are in use. Once a client registers with card services, it will then be informed of relevant events pertaining to the PC cards currently installed.

The PC4500/4800 is accessed via 32 contiguous 16-bit I/O registers. Each register is assigned an offset which relates to the base port address used by the socket. (Base port address plus offset yields the port address through which each register is accessed.) Since all registers are 16-bit registers, two I/O ports are used for register access. The hardware will transfer 8 bits via the respective port address and 8 bits via port address+1. Therefore, an area of 64 continuous I/O ports are required for PC4500/4800 controller access.

All sample code shown in this chapter uses “PC4500” as a part of variable names and in comments. This sample code, however, is appropriate for use with the PC4800 as well.

Host Memory Map

The host has two memory spaces available to it: I/O Space and Attribute Memory. Common memory is not available on the PC4500/4800.

Host Attribute Memory

The host attribute memory consists of the CIS and the attribute memory registers. The complete PC4500/4800 CIS description can be found in Appendix B.

The CIS is copied from the flash to the RAM during power on reset. (The READY signal remains deasserted until the copy process is completed.)

Table 7.1 – Card Configuration Information

Offset	Description
000H-300H	Card Information Structure (CIS)
3E0H	Configuration Option Register must set to 0x45 to enable Host control (I/O) Registers.
3E2H	Card Configuration and Status Register
3E4H	Pin Replacement Register
3E6H	Socket and Copy Register (not used)

Host Common Memory

There is no common memory accessible on the PC4500/4800.

Host I/O Registers

The host has 32 consecutive 16-bit I/O registers for accessing the PC4500/4800 card. Table 7-2 lists the register assignments according to functionality.

All registers (except for the Data0, Data1 and AuxData) can be read without restriction since the read operation will not impact the PC4500/4800 protocol processor. The exceptions require that an internal data pointer be affected.

All registers are 16-bit registers, therefore, it is assumed that the host software writes or reads 16 bits per I/O cycle. If the hardware environment is only capable of transferring 8 bits per I/O cycle, the register word should be transferred as follows: first the low-order byte (bits 0-7) should be accessed via the I/O register offset (even), followed by the high-order byte (bits 8-15) via register offset+1 (odd). **To prevent undefined processor behavior, both bytes must be transferred before any other access is allowed (includes another register access as well as an attribute memory access).**

The following is sample code for accessing the PC4500/4800 I/O registers:

```
unsigned short      Base4500IO;                                                 // I/O base address
#if IO_IS_16BIT
                                                                                // I/O is 16-bit
    #define         OUT4500(register, u16value)     outportw(Base4500IO+register, u16value)
    #define         IN4500(register)                inportw(Base4500IO+register)
#else
                                                                                // I/O is 8-bit
    #define         OUT4500(register, u16value)     out4500_8bit(register, u16value)
    #define         IN4500(register)                in4500_8bit(register)

    void out4500_8bit(unsigned short register, unsigned short u16value)
    {
        push_interrupt_enable_state();                                          // save interrupt enable state
        disable_interrupts();                                                   // disable interrupts
        outportb(Base4500IO+register+0, (char)(u16value & 0xFF));
        outportb(Base4500IO+register+1, (char)(u16value >> 8));
        pop_interrupt_enable_state;                                             // restore interrupt enable state
    }

    unsigned short in4500_8bit(unsigned short register)
    {
        unsigned short u16RetVal;
        push_interrupt_enable_state();                                          // save interrupt enable state
        disable_interrupts();                                                   // disable interrupts
        u16RetVal = inportb(Base4500IO+register+0);
        u16RetVal += ((u16)inportb(Base4500IO+register+1)) << 8;
        pop_interrupt_enable_state;                                             // restore interrupt enable state
    }
#endif
```

Table 7.2 – PC4500/4800 Register Summary

Command/Status			Events	
0x00	Command		0x30	EvStat
0x02	Param0		0x32	EvIntEn
0x04	Param1		0x34	EvAck
0x06	Param2		Host Software	
0x08	Status		0x28	SWSupport0
0x0A	Resp0		0x2A	SWSupport1
0x0C	Resp1		0x2C	SWSupport2
0x0E	Resp2		0x2E	SWSupport3
Memory (FID/RID) Access			FID Management	
Buffer Access Pointer 0 (BAP0)			0x20	RxFID
0x18	Select0		0x22	TxAllocFID
0x1C	Offset0		0x24	TxComplFID
0x36	Data0		Control	
Buffer Access Pointer 1 (BAP1)			0x10	LinkStatus
0x1A	Select1		Auxiliary Port	
0x1E	Offset1		0x3A	AuxPage
0x38	Data1		0x3C	AuxOffset
			0x3E	AuxData

Note: all other registers are reserved

Recognizing the PC4500/4800

The PC4500 and PC4800 cards can be recognized via the Manufacturer Codes in the Card Information Structure (CIS) contained in attribute memory. The values are:

Manufacturer Code
PC4500: 0x015F
PC4800: 0x015F

Product Code
PC4500: 0x0005
PC4800: 0x0007

Card services provides a GET_CONFIGURATION_INFO function which returns the manufacturer and product codes from a socket. Programmers writing their own host software should process the CIS to verify that the card is a PC4500 or PC4800.

Bootstrap -- Starting the PC4500/4800

The following is the correct start sequence from the host perspective.

1. Wait for Card Detect signals to go active.
2. Enable power (VCC) to the socket. Also, set VPP1, VPP2 = 5V.
3. Delay for power to stabilize (200 ms).
4. Release PCMCIA reset.
5. Delay at least 10us.
6. Wait for Card Ready to go active.
7. Read the CIS to identify card.
8. Configure socket to have an I/O window with 32 consecutive 16-bit ports.
9. Set Card Configuration Options Register to 0x45. [Currently this is at address 0x3E0 of attribute memory; but, use the CIS to determine the correct address]
10. Release PC4500/4800 from reset by issuing a NOP command -- OUT4500(COMMAND, 0x0010).
11. Wait for EvStat.Cmd bit to be set.
12. Ack the command completion by setting EvAck.Cmd.

Resetting the PC4500/4800

Three methods exist to reset the PC4500/4800:

1. use the PCMCIA reset line,
2. by writing a 0x80 into the Configuration Options Register, followed by writing a 0x00 into the same register (after the initial boot).
3. by issuing the restart command.

Note, when resetting the card, the VPP1 and VPP2 lines must be held at 5V for normal operation. Startup is then the same as for normal bootstrap of the card.

Interrupt Service Routine (ISR) processing

The correct interrupt service routine processing is as follows:

1.Disable the interrupt source on the host (mask the PIC).
2.Prepare the host for another interrupt (EOI the PIC).
3.Determine the active interrupts:

active_interrupts = IN4500(EVSTAT);

Note: even masked interrupts will be active.

4.Process the active interrupts. (See Event Handling for details).

5.Acknowledge the interrupts and disable the interrupt:

OUT4500(EVACK, active_interrupts);

6.Disable interrupts
Note: Disabling and re-enabling the interrupt may be required to ensure that an edge is generated for the next interrupt:

SaveIntEn = IN4500(INTEN);
OUT4500(INTEN, 0x0000);

7.Unmask the interrupt on the host.
8.Reenable the interrupt:

OUT4500(INTEN, SaveIntEn);

Note: It is important to process the interrupt source before acknowledging, since the acknowledge may result in loss of information associated with the interrupt source.

Enabling the PC4500/4800

Enabling the PC4500/4800 for operation consists of the following steps:

1. recognizing the card
2. boot strapping the PC4500/4800
3. installing an interrupt service routine
4. optionally pre-allocating transmit fids
5. writing the configuration rids
6. enabling the MAC

After bootstrapping, the following is sample code for configuring and enabling the MAC:

```
tdsCommand          Cmd;                    // see following section for definition
tdsResponse         Rsp;                    // see following section for definition
PC4500_CONFIG       cfg;                    // see following section for definition

typedef struct {
    unsigned short  SsidLen;
    unsigned char   Ssid[32];
} PC4500_SSID;

struct MySsid {
    u16             u16RidLen;
    PC4500_SSID     aSSID[2];
    } = {
                    0,                      // PC4500 will ensure that the RidLen is unchanged
                    { 6, "MYSSID" }         // SSID length and value
                    { 0, "" }               // zero length SSID to end the list
}

    u16 minimum_mac_enable(void)
    {
        memset(Cmd, 0, sizeof(Cmd));
        Cmd.Command = MAC_DISABLE;                          // disable in case already enabled
        issuecommand(&Cmd, &Rsp);
                                                            // general configuration
                                                            (read/modify/write)
        PC4500_readrid(0xFF10, &cfg, sizeof(cfg));
        cfg.u16OperatingMode = MODE_STA_ESS;                // station in ESS mode
        PC4500_writerid(0xFF10, &cfg, sizeof(cfg));
                                                            // ssid list
        PC4500_writerid(0xFF11, &MySsid, sizeof(MySsid));
        memset(Cmd, 0, sizeof(Cmd));
        Cmd.Command = MAC_ENABLE;                           // enable the MAC
        issuecommand(&Cmd, &Rsp);

        if ((Rsp.Status & 0xFF00) != 0) {
            Reason      = Rsp.Resp0;
            BadRidNumber = Rsp.Resp1;
            BadRidOffset = Rsp.Resp2;
        }
        return SUCCESS;
    }
```

Command and Status Register Descriptions

The PC4500/4800 provides 4 I/O registers for issuing commands to the protocol processor and 4 I/O registers for status response. These registers provide a command / status interface for sequential protocol processor control functions. The command code and related information is entered in the Command and Param registers. The process is executed by the PC4500/4800 and then the resulting status is provided in the Status and Resp registers. Completion of a command is signaled and acknowledged using the Event registers.

The command response mechanism provides semaphores on the command/response to prevent the host from overwriting a command that has not yet been processed and to prevent the PC4500/4800 from overwriting a response. The mechanism allows the host to issue a new command after the PC4500/4800 has acknowledged, but is still processing the previous command. Note, a command may be stalled in the Command/Param registers until a previous response is acknowledged.

The sequence for issuing a command to the PC4500/4800 is as follows:

1. wait for Command.Busy to be clear
2. write the required parameters (Param0, Param1, Param2)
3. write the command (Command)
4. wait for EvStat.Cmd to indicate that the command has completed
5. read the completion status and responses (Status, Resp0/1/2)
6. acknowledge the command by setting EvAck.Cmd

When the command completes, the Status register will contain the command code in the lower byte. If the command completed successfully, the upper byte of the Status will be zero, and Resp0, Resp1 and Resp2 will contain responses that change depending on the command.

If the command failed, the upper byte of the Status Register will be non-zero, and Resp0 will contain the error code. Additional information may be conveyed in Resp1 and Resp2 depending on the command.

Some commands allow for successful completion to not be indicated to the host. This is accomplished by setting the NoResponse bit when writing the Command Register. If the command successfully executes, there will be no EvStat.Cmd event. If the command fails, a response will be returned as normal.

If commands are to be issued from multiple contexts (e.g. interrupt routines) then interrupts should be disabled as appropriate to prevent a context switch from causing improper operation:

1. wait for Command.Busy to be clear
2. disable interrupts;
3. if Command.Busy is set, reenable interrupts and return to step 1
4. write the required parameters (Param0, Param1, Param2)
5. write the command (Command)
6. reenable interrupts;
7. .... as above ...

A command will stall in the command register (Command.Busy won't clear) if a previous command completion (EvStat.Cmd is set) has not been acknowledged. Setting EvAck.Cmd will acknowledge the previously unacknowledged command and allow the firmware to proceed with the new command.

In some circumstances, a command may be issued that is not processed by the PC4500/4800. The PC4500/4800 will always zero the Command register after accepting the command. If the Command register is read back with the Command.Busy bit cleared and a non-zero value, then the command should be reissued by writing the Command register again.

In some circumstances Command.Busy also may not clear. This may occur when the host reads the Command register at the same time as the firmware attempts to clear the Command.Busy bit.

A work-around is available for clearing a stuck Command.Busy, by setting EvAck.ClearCommandBusy. If the firmware is not processing a command, the Command.Busy bit will be cleared.

The following is an example for issuing a single command:

```
typedef struct {
    unsigned short    Command;
    unsigned short    Param0;
    unsigned short    Param1;
    unsigned short    Param2;
} tdsCommand;

typedef struct {
    unsigned short    Status;
    unsigned short    Resp0;
    unsigned short    Resp1;
    unsigned short    Resp2;
} tdsResponse;

unsigned short issuecommand(tdsCommand *pCmd, tdsResponse *pRsp) {
                                                    // wait for command.busy to clear normally
                                                    // we will only issue one command at a time so no need
    OUT4500(PARAM0,  pCmd->Param0);
    OUT4500(PARAM1,  pCmd->Param1);
    OUT4500(PARAM2,  pCmd->Param2);
    OUT4500(COMMAND, pCmd->Command);
    while ( (IN4500(EVSTAT) & EV_CMD) == 0) {
        if (IN4500(COMMAND) == command) {           // PC4500 didn't notice command, try again
            OUT4500(COMMAND, command);
        }
    }
                                                    // command completed
    pRsp->Status = IN4500(STATUS);
    pRsp->Resp0  = IN4500(RESP0);
    pRsp->Resp1  = IN4500(RESP1);
    pRsp->Resp2  = IN4500(RESP2);
                                                    // clear stuck command busy if necessary
    if (IN4500(COMMAND) & COMMAND_BUSY) {
        OUT4500(EVACK, EV_CLEARCOMMANDBUSY);
    }
                                                    // acknowledge processing the status/response
    OUT4500(EVACK, EV_CMD);
}
```

```
typedef struct {
    unsigned short  u16Len;                                 /* sizeof(PC4500_CONFIG) */
    unsigned short  u16OperatingMode;                       /* operating mode       */

    #define         MODE_STA_IBSS               0
    #define         MODE_STA_ESS                1
    #define         MODE_AP                     2
    #define         MODE_AP_RPTR                3
    #define         MODE_ETHERNET_HOST          (0<<8)      /* rx payloads converted */
    #define         MODE_LLC_HOST               (1<<8)      /* rx payloads left as is */
    #define         MODE_AIRONET_EXTEND         (1<<9)      /* enable Aironet extenstions */
    #define         MODE_AP_INTERFACE           (1<<10)     /* enable ap interface extensions */
    unsigned short  u16ReceiveMode;                         /* receive mode */
    #define         RXMODE_BC_MC_ADDR           0
    #define         RXMODE_BC_ADDR              1           /* ignore multicasts */
    #define         RXMODE_ADDR                 2           /* ignore multicast and broadcast */
    #define         RXMODE_RFMON                3           /* wireless monitor mode */
    #define         RXMODE_RFMON_ANYBSS         4
    #define         RXMODE_LANMON               5           /* lan style monitor -- data packets only */
    #define         RXMODE_DISABLE_802_3_       (1<<8)      /* disables 802.3 header on rx */
                    HEADER
    unsigned short  u16FragmentThreshold;
    unsigned short  u16RtsThreshold;
    unsigned char   au8StationMacAddress[6];
    unsigned char   au8Rates[8];
    unsigned short  u16ShortRetryLimit;
    unsigned short  u16LongRetryLimit;
    unsigned short  u16TxLifetime;                          /* in kusec */
    unsigned short  u16RxLifetime;                          /* in kusec */
    unsigned short  u16Stationary;
    unsigned short  u16Ordering;
    unsigned short  u16DeviceType;                          /* for overriding device type */
    unsigned short  _reserved1[5];
                                                            /*---------- Scanning/Associating ----------*/
    unsigned short  u16ScanMode;
    #define         SCANMODE_ACTIVE             0
    #define         SCANMODE_PASSIVE            1
    #define         SCANMODE_AIROSCAN           2
    unsigned short  u16ProbeDelay;                          /* in kusec */
    unsigned short  u16ProbeEnergyTimeout;                  /* in kusec */
    unsigned short  u16ProbeResponseTimeout;
    unsigned short  u16BeaconListenTimeout;
    unsigned short  u16JoinNetTimeout;
    unsigned short  u16AuthenticationTimeout;
    unsigned short  u16AuthenticationType;
    #define         AUTH_OPEN                   1
    #define         AUTH_SHAREDKEY              2
    #define         AUTH_EXCLUDENONWEP          4
    unsigned short  u16AssociationTimeout;
    unsigned short  u16SpecifiedApTimeout;
    unsigned short  u16OfflineScanInterval;
    unsigned short  u16OfflineScanDuration;
    unsigned short  u16LinkLossDelay;
    unsigned short  u16MaxBeaconLostTime;
    unsigned short  u16RefreshInterval;
```

```
    #define         DISABLE_REFRESH         0xFFFF
    unsigned short  _reserved1a[1];
                                            /*---------- Power save operation ----------*/
    unsigned short  u16PowerSaveMode;
    #define         POWERSAVE_CAM           0
    #define         POWERSAVE_PSP           1
    #define         POWERSAVE_PSP_CAM       2
    unsigned short  u16SleepForDtims;
    unsigned short  u16ListenInterval;
    unsigned short  u16FastListenInterval;
    unsigned short  u16ListenDecay;
    unsigned short  u16FastListenDelay;
    unsigned short  _reserved2[2];
                                            /*---------- Ap/Ibss config items ----------*/
    unsigned short  u16BeaconPeriod;
    unsigned short  u16AtimDuration;
    unsigned short  u16HopPeriod;
    unsigned short  u16ChannelSet;
    unsigned short  u16Channel;
    unsigned short  u16DtimPeriod;
    unsigned short  _reserved3[2];
                                            /*---------- Radio configuration ----------*/
    unsigned short  u16RadioType;
    #define         RADIOTYPE_DEFAULT       0
    #define         RADIOTYPE_802_11        1
    #define         RADIOTYPE_LEGACY        2
    unsigned char   u8RxDiversity;
    unsigned char   u8TxDiversity;
    unsigned short  u16TxPower;
    #define         TXPOWER_DEFAULT         0
    unsigned short  u16RssiThreshold;
    #define         RSSI_DEFAULT            0
    unsigned short  u16RadioSpecific[4];
                                            /*---------- Aironet Extensions ----------*/
    unsigned char   au8NodeName[16];
    unsigned short  u16ArlThreshold;
    unsigned short  u16ArlDecay;
    unsigned short  u16ArlDelay;
    unsigned short  _reserved4[1];
                                            /*---------- Aironet Extensions ----------*/
    unsigned short  u16MagicAction;
    #define         MAGIC_ACTION_STSCHG     1
    #define         MACIC_ACTION_RESUME     2
    #define         MAGIC_IGNORE_MCAST      (1<<8)
    #define         MAGIC_IGNORE_BCAST      (1<<9)
    #define         MAGIC_SWITCH_TO_PSP     (0<<10)
    #define         MAGIC_STAY_IN_CAM       (1<<10)
}
PC4500_CONFIG;
```

Command Register (I/O offset 0x00)

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	busy	Additional Info							No Ack	0	Command Code					

This register is used to write commands. Busy is automatically set to 1 when a value is written to this register (irrespective of the actual value written) and reset to 0 when the command is accepted.
A command written while the Busy bit is set to 1 will be neglected.

The commands are described in detail in later sections.

COMMAND SUMMARY	
Command Number	**Command**
0x0000	NOP
0x0001	Enable
0x0002	Disable
0x0003	Force a Loss of Sync
0x0004	Firmware Restart (soft reset)
0x0005	Host Sleep (must be issued as 0x0085)
0x0006	Magic Packet
0x0008	Read the Configuration from nonvolatile storage
0x000A	Allocate Transmit Buffer
0x000B	Transmit
0x000C	Deallocate
0x0010	NOP (same as 0x0000)
0x0021	Access RID
0x0028	Allocate Buffer
0x0030	PSP nodes (AP only)
0x003E	Set PHY register
0x003F	Transmitter Test
0x0085	Go to Sleep (No Ack bit is mandatory)
0x0108	Save the configuration to nonvolatile storage

NoAck – An EvStat.Cmd will not be generated when the command is complete. (This option is not supported by all commands).

Param0-2 Registers (I/O offsets 0x02 – 0x06)

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	Command Parameter															

These registers are used to write required parameters for various commands.
Command parameters should only be written when the Command.Busy bit is 0. Therefore, parameters must be entered before the command is entered in the Command register.

Details of specific parameters are provided along with the command descriptions.

Note: Although a command (and parameters) may be written to the Command and Param registers when Command.Busy is 0, the command will not be processed by the PC4500/4800 until the previous command is acknowledged via EvAck.Cmd event.

Status Register (I/O offset 0x08)

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	0	Result							0	0	Command Code					

This register is used to read the status resulting from a command execution.

Status availability is signaled by the EvStat.Cmd bit. Future commands are only accepted (and hence, subsequent status can become available) after the current status is acknowledged with the EvAck.Cmd bit.

Details of specific status information is provided with the appropriate command descriptions.

Resp0-2 Registers (I/O offsets 0x0A - 0x0E)

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	Command Response															

These registers are used to read the response resulting from various command executions. The register values are only valid while status availability is signaled (before status acknowledgement).

Details of specific response information is provided with the appropriate command descriptions.

Event Register Descriptions

These registers provide for notification from the PC4500/4800 to the host. Each event is individually acknowledgeable, individually maskable and each may generate an interrupt signal. Each specific event is signaled in one of the EvStat register bits and acknowledged with the corresponding bit in the EvAck register. Additionally, when the corresponding EvIntEn register bit is set, event signaling generates a hardware interrupt condition.

Each event has associated registers that must be read before the event is acknowledged or the information being indicated by the event may be lost. The following table shows the associated register for each event.

EvStat Register (I/O offset 0x30)

The EvStat register is used to read if one or more events have occurred. This a read-only register.

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	0	0	0	0	0	0	0	Awake	Link	0	0	cmd	alloc	Txexc	TX	RX

The event descriptions related to the register bits are:

Event	Associated Registers	Description
Awake	None – PSP mode	This bit is used only in power save mode. Awake event is issued in response to an EvAck.Awaken. To allow for maximal power saving, the host must inform the PC4500/4800 that the host will no longer access the PC4500/4800. To leave this mode, the host must issue an EvAck.Awaken and then wait for EvStat.Awake. After EvStat.Awake, the PC4500/4800 is normally accessible.
Link	LinkStatus	Indicates that a LinkStatus word is available. Acknowledge allows the PC4500/4800 to issue the next link status. Read LinkStatus before acknowledging.
Cmd	Status Resp0 Resp1 Resp2	Indicates that a command has completed and that its completion response is available . Acknowledge allows PC4500/4800 to issue next response. For transmit, this only indicates that the transmission has been queued, a subsequent EvStat.Tx or EvStat.TxExc will indicate the success or failure.
Alloc	TxAllocFid	Indicates that a TxAllocFID is available (Alloc command completed). Acknowledge allows PC4500/4800 to issue next Alloc command; Read TxAllocFID before acknowledging.
TxExc	TxComplFid	Indicates that a TxComplFID is available and that the transmission was unsuccessful. Read the FID and access the status (cause of failure) before acknowledging. Acknowledging allows the PC4500/4800 to issue the next TxComplFid. Acknowledging will release the TxComplFid to the free pool or return the FID to host control (if TxControl.NoRelease was set).
TX	TxComplFid	Indicates that a TxComplFID is available and that the transmission was successful. Read the FID and access the status (cause of failure) before acknowledging. Acknowledging allows the PC4500/4800 to issue the next TxComplFid. Acknowledging will release the TxComplFid to the free pool or return the FID to host control (if TxControl.NoRelease was set).
RX	RxFid	Indicates that a RxFID is available Acknowledge releases the RxFID to the free pool. Read the FID before acknowledging.

Additional information:

Link – occurs when the connection control status has changed and the LinkStat register contains related status qualification information. This event occurs again if subsequent connection control status information is already available when the current event occurrence is acknowledged.

Each event may occur again after being acknowledged if another occurrence of the event is pending.

An event should only be acknowledged after the relevant information has been processed. Acknowledging the event first may result in the associated information being purged before the host is able to access it.

The EvAck.ClearCommandBusy provides a mechanism for clearing a stuck command busy bit. A hardware errata will result in the Command.Busy bit not being cleared if the PC4500/4800 attempts to clear the Command.Busy at the same time as the host reads the Command register. The result is that no further commands can ever be issued. A work-around is available by setting the EvAck.ClearCommandBusy. After receiving this, the PC4500/4800 firmware will again attempt to clear the Command.Busy bit
(provided that it isn't actually processing a command).

The EvAck.Awaken provides a mechanism to awaken the card when in PowerSave mode. When the card is in maximum sleep mode, most registers are unavailable.
Two writes, back-to-back, are required to awaken the PC4500/4800 -- the first will awaken the hardware, the second will actually issue the EvAck.Awaken.

EvIntEn Register (I/O offset 0x32)

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	0	0	0	0	0	0	0	0	Link En	0	0	Cmd En	alloc En	Txexc En	TX En	RX En

The EvIntEn register is used to write a control mask for hardware interrupt generation by events. For each mask bit that is set to 1, the hardware interrupt condition is generated when the corresponding bit (same position) of the EvStat register becomes 1.

EvAck Register (I/O offset 0x34)

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	0	CSB	WakeRequest	0	0	0	0	HasAwoken	Link	0	0	cmd	alloc	Txexc	TX	RX

The EvAck register is used to write event acknowledgements. For each bit that is set to 1, event occurrence as related to the corresponding bit (same position) of the EvStat register is acknowledged. Notice that an event occurrence always needs to be acknowledged.

Note: Values read from this register are undefined. Acknowledgement bits are internally set to 0 by the PC4500/4800 controller upon acknowledgment completion. Hence, only register bits set to 1 affect the behavior of the PC4500/4800 controller.

Some ACKs used in this register are actually used as attention signals to the PC4500/4800.

CSB – (Clear Stuck Busy) is used as an indicator to the firmware to clear a stuck command busy bit due to hardware problems.

WakeUpRequest – is used to awaken the card from PSP sleep mode. Refer to the section on power save operation for more details.

Basic FID Access

Memory items consist of:

FIDs - Frame Identifiers (transmit/receive packets or allocated memory)
RIDs - Resource Identifiers (configuration items)

FID/RID are accessed using three I/O registers: selector, offset, data.

The first register is the selector register, which chooses the desired FID/RID. The second register is the offset register, which selects the position within the FID/RID. The third register is the data register which allows reads and writes to the FID/RID. Successive reads and writes go to sequential locations within the FID/RID.

Valid FIDS are obtained from the TxAllocFid, TxComplFID, or RxFID I/O-registers, or possibly in response to a successful command.

A RID (Resource Identifier) may also be used in the Selector fields. These are predefined selectors from 0xFF00 to 0xFFFF. The RIDs are used to access configuration items.

Additional information on FID access is provided in the section on "Memory (FID/RID) Access".

FID Management Register Descriptions

These registers provide the Frame Identifier (FID) of the PC4500/4800 buffers that become available as result of an asynchronous process. A specific FID value is only valid when read while availability is signaled and before that availability is acknowledged. In other words, you must read and process RxFID (including reading it) before issuing EvAckRx.

RxFID Register (I/O offset 0x20)

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	Received FID															

This register is used to read the FID of received frame structure buffers. This FID becomes available as a result of an asynchronous packet reception. The received FID availability is signaled by the EvStat.Rx bit. A subsequent received FID can only become available after availability of the current FID is acknowledged with EvAck.Rx bit.

TxAllocFID Register (I/O offset 0x22)

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	Allocated Transmit FID															

This register is used to read the FID of transmit frame structure buffer. This FID becomes available as a result of the asynchronous allocation process initiated by an Allocate command. Allocated transmit FID availability is signaled by the EvStat.Alloc bit. A subsequent allocated transmit FID can only become available after availability of the current FID is acknowledged with EvAck.Alloc bit.

TxComplFID Register (I/O offset 0x24)

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	Completed Transmit FID															

This register is used to read the FID of transmit frame structure buffers. This FID becomes available as a result of a completion (or failure) of an asynchronous transmission initiated by a Transmit command. Completed transmit FID availability is signaled by the EvStat.Tx or the EvStat.TxExc bits. A subsequent completed transmit FID can only become available after availability of the current FID is acknowledged with the EvAck.Tx or the EvAck.TxExc bits.

Buffer Access Register Descriptions

These two sets of registers (BAP0 and BAP1) are used for PC4500/4800 buffer access. The specific buffer is indicated by a FID or RID value in the Select register. The first word of the specific area within that buffer to be accessed is indicated in the Offset register. After interpretation of those addressing values by the PC4500/4800 controller, consecutive buffer words can be accessed by repeatedly writing or reading the Data register. It is not possible to read the same work directly after it is written (the Selector / Offset register must be written again).

Note: When another area of the same buffer needs to be accessed, both the Select and the Data offset registers must be written since the PC4500/4800 controller zeros the Select Register.

Select0-1 Registers (I/O offsets 0x18 or 0x1A)

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	FID / RID															

This register is used to write the FID/RID of a buffer to be accessed through the related Data register. A valid FID/RID value must be written while Busy of the related Offset register is 0 and before the Data offset value is written into that Offset register. A valid FID value is obtained via the FID management registers. A valid RID value is 0xFF??, where ?? can be any value. This RID value always indicates the temporary record buffer (see Access command).

Offset0-1 Registers (I/O offsets 0x1C or 0x1E)

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	busy	Err	done	0	Data offset											0

This register is used to write the initial buffer offset and to read the buffer access status. A Data offset value must be written while Busy is 0 and after the FID / RID value is written into the related Select register.

Busy – is automatically set to 1 when a value is written into this register (irrespective of the actual bit value written) and is reset to 0 when the Err field becomes valid.

Err – 0 indicates that the data can be accessed.
1 indicates that the given offset points outside the buffer boundary or the FID / RID in the related Select register is incorrect.

Done – the done bit will become 1 to indicate that the request was processed. (If done does not get set, then restart.)

Data0-1 Registers (I/O offsets 0x36 or 0x38)

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	Data															

This register is used to write or read buffer data. The specific buffer and buffer word initially accessed is determined by values previously written to the related Select and Offset registers. When a word is written to or read from this register, the internal data pointer is automatically incremented. Therefore, repeated Data register access addresses consecutive buffer words.
Initial access to the Data register is only allowed if Busy of the related Offset register is 0. During subsequent Data register accesses (eg. without writes to the related Select and Offset registers) Busy remains 0.

Alternating writes to and reads from this register is NOT recommended.

Note: due to the automatic internal data pointer increments, the Data offset needs to be rewritten into the related Offset register when repeated accesses to the same buffer word are needed.

The following is the correct method for accessing memory (FIDs):

```
SelectX = FIDvalue; // select the appropriate FID/RID
OffsetX = offset; // desired offset
while (OffsetX.Busy) ; // -- add applicable timeout here
if (OffsetX.Err) { // invalid FID or offset
  error in access;
  return ERROR;
  };
read/write to DataX; // note, auto-increments to next word
```

Notes:
The selector must be written before the offset.
The least-significant bit of the offset register is zero since only words are accessible.
Only word writes are available, a read/modify write is required to change a byte. In this case the offset register would have to be rewritten since it would auto-increment to the next address.
The OffsetX register is not updated by reads/writes to the DataX register; the original value remains in the register.
Note, since the OffsetX register is not updated, an interrupt service routine cannot use the same BAPx registers as are in use by an interrupted routine.

Sample C-code

```
bap0_setup(u16 rid, u16 offset)                  // this example uses SELECT0/OFFSET0/DATA0
{
    OUT4500(SELECT0, rid);
    OUT4500(OFFSET0, offset);
    while (1) {
        status = IN4500(OFFSET0);
        if (status & BAP_BUSY) {
            if (timeout) {                       // suggested timeout of 500 usec minimum
                push_interrupt_enable_state();   // save interrupt enable state
                disable_interrupts();            // disable interrupts
                OUT4500(SELECT0, rid);
                OUT4500(OFFSET0, offset);
                pop_interrupt_enable_state();    // restore interrupt enable state
                restart_timeout();
            }
            continue;
        }
        if (status & BAP_ERR) {                  // invalid rid or offset
            return ERROR;
        }
        if (status & BAP_DONE) {                 // success
            return SUCCESS;
        }
                                                 // -- PC4500 missed it, try again
        OUT4500(SELECT0, rid);
        OUT4500(OFFSET0, offset);
    }
}

                                                 // requires call to bap0_setup() first
bap0_read(u16 *pu16Dst, int bytelen)
{
    bytelen = (bytelen + 1) & (~1);              // round up to even value
    while (bytelen > 0) {                        // each access to DATA0 will auto-increment
        *pu16Dst++ = IN4500(DATA0);              // to the next word
        bytelen -= 2;
    }
    return SUCCESS;
}
                                                 // requires call to bap0_setup() first
bap0_write(u16 *pu16Src, int bytelen)
{
    bytelen = (bytelen + 1) & (~1);              // round up to even value
    while (bytelen > 0) {                        // each access to DATA0 will auto-increment
        OUT4500(DATA0, *pu16Src++);              // to the next word
        bytelen -= 2;
    }
    return SUCCESS;
}
```

LinkStat Register (I/O offset 0x10)

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	Loss sync	0	0	0	0	0	0	0	0	0	0	0	EventCode			

This register is used to read information that qualifies a Link event generation. Link events are generated by an asynchronous connection control process initiated by the Enable command. Link event information availability is signaled by the Link bit in the EvStat register. Subsequent Link event information can only become available after availability of the current Link event information is acknowledged with the EvAck.LinkAck bit.

The following are the LinkStatus values returned by the PC4500/4800.

0x8000	Loss of sync -- missed beacons
0x8001	Loss of sync -- max retries
0x8002	Loss of sync -- average retry level (ARL) exceeded
0x8003	Loss of sync -- host request
0x8004	Loss of sync -- TSF synchronization
0x81xx	Deauthentication (reason code in low byte)
0x82xx	Disassocation (reason code in low byte) Possibly due to AP max retrying on a packet to station.
0x84xx	Association failed (status code in low byte)
0x03xx	Authentication failure (status code in low byte)
0x0400	Associated

Note, the upper bit of Link Status is set if the station is not associated.

The following reason codes are defined by the IEEE 802.11 standard:

Reason Code	**Meaning**
0	Reserved
1	Unspecified Reason
2	Previous Authentication no longer valid
3	Deauthenticated because sending station is leaving (has left) IBSS or ESS
4	Disassociated due to inactivity
5	Disassociated because AP is unable to handle all currently associated stations
6	Class 2 frame received from non-Authenticated station
7	Class 3 frame received from non–Associated station
8	Disassociated because sending station is leaving (has left) BSS
9	Station requesting (Re)Association is not Authenticated with responding station
10 - 65535	Reserved

Note: For IBSS mode, a Deauthentication notification may not have the 0x8000 bit set.

Host Software Registers

These registers are available for storage of host software values. The register values are not changed or interpreted by the PC4500/4800 controller.

SwSupport0-3 Registers (I/O offsets 0x28 - 0x2E)

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	SW support value															

Host Auxiliary Registers

These registers are available for AP host support. (Refer to the section on AP specific information for more details.)

AuxPage Register (I/O offset 0x3A)

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	Page selection															

AuxOffset Register (I/O offset 0x3C)

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	Offset value															

AuxData Register (I/O offset 0x3E)

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	Data value															

Command Descriptions

COMMAND SUMMARY	
Command Number	**Command**
0x0000	NOP
0x0001	Enable
0x0002	Disable
0x0003	Force a Loss of Sync
0x0004	Firmware Restart (soft reset)
0x0005	Go to Sleep (must be issued as 0x0085)
0x0006	Magic Packet
0x0008	Read the configuration from nonvolatile storage
0x000A	Allocate Transmit Buffer
0x000B	Transmit
0x000C	Deallocate
0x0010	NOP (same as 0x0000)
0x0021	Access RID
0x0028	Allocate Buffer
0x0030	PSP nodes (AP only)
0x003E	Set PHY register
0x003F	Transmitter Test
0x0085	Go to Sleep (No Ack bit is mandatory)
0x0108	Save the configuration to nonvolatile storage

Note: In the following tables, the Resp0,1,2 fields are shown as zeroes for successful command responses. These fields are not guaranteed to be zero and should be considered as don't cares since the significance is found with the successful Status entry.

NOP Command

Note, the NOP command should never fail, it is provided as verification that the firmware is operational.

NOP Command				
Command (0x00)	Param0 (0x02)	Param1 (0x04)	Param2 (0x06)	
0x0000 or 0x0010	0x0000	0x0000	0x0000	NOP command
NOP Responses – Success				
Status (0x08)	Resp0 (0x0A)	Resp1 (0x0C)	Resp2 (0x0E)	Description
0x0000 or 0x0010	0x0000	0x0000	0x0000	Successful NOP

Enable Command

The Enable command is used to start operation of the card after writing the configuration information.

Enable Command				
Command (0x00)	Param0 (0x02)	Param1 (0x04)	Param2 (0x06)	
0x0001	0x0000	0x0000	0x0000	Basic Enable This option enables both the MAC and receive operations
0x0101	0x0000	0x0000	0x0000	Enable Only the MAC Allows the PC4500/4800 to associate to an AP, but no received packets will be passed to the host. (Required for NDIS3 which cannot install interrupt handler until long after the card has been configured and started.)
0x0201	0x0000	0x0000	0x0000	Enable Receive
Enable Responses = Success				
Status (0x08)	Resp0 (0x0A)	Resp1 (0x0C)	Resp2 (0x0E)	Description
0x0001	0x0000	0x0000	0x0000	Successful Enable command
Enable Responses = Failure				
Status (0x08)	Resp0 (0x0A)	Resp1 (0x0C)	Resp2 (0x0E)	Reason
	Error Qualifier	RID	RID Offset	
0x7F01	0x0002	0x0000	0x0000	Illegal format (Command)
0x7F01	0x0006	0x0000	0x0000	Is enabled already
0x7F01	0x0003	….	…..	Invalid RID as follows
0x7F01	0x0080	0xFF10	0x0002	Invalid Mode
0x7F01	0x0083	0xFF10	0x0004	Invalid Receive Mode
0x7F01	0x0084	0xFF10	0x000A	Invalid Mac Address
0x7F01	0x0086	0xFF10	0x0022	Invalid Ordering Selection
0x7F01	0x0087	0xFF10	0x0030	Invalid ScanMode
0x7F01	0x0088	0xFF10	0x003E	Invalid Authentication Type
0x7F01	0x0089	0xFF10	0x0050	Invalid Power Save Mode
0x7F01	0x0082	0xFF10	0x0060	Invalid Beacon Interval
0x7F01	0x0081	0xFF10	0x0064	Invalid Hop Interval
0x7F01	0x008A	0xFF10	0x0070	Invalid Radio Type
0x7F01	0x008B	0xFF10	0x0072	Invalid Diversity
0x7F01	0x008C	0xFF11	XXXX	Invalid SSID list entry
0x7F01	0x008D	0xFF12	XXXX	Invalid Specified AP entry

Disable Command

The disable command is used to disable the operation of the card, typically to allow for a change of the configuration and subsequent reenable.

Disable Command				
Command (0x00)	Param0 (0x02)	Param1 (0x04)	Param2 (0x06)	
0x0002	0x0000	0x0000	0x0000	Disable the MAC
Disable Responses = Success				
Status (0x08)	Resp0 (0x0A)	Resp1 (0x0C)	Resp2 (0x0E)	Description
0x0002	0x0000	0x0000	0x0000	Successful Disable command
Disable Responses = Failure				
Status (0x08)	Resp0 (0x0A)	Resp1 (0x0C)	Resp2 (0x0E)	Reason
	Error Qualifier			
0x7F02	0x0002	0x0000	0x0000	Illegal format (Command)
0x7F02	0x0000	0x0000	0x0000	MAC is already disabled

Lose Sync Command

This command is used by a station or a repeater to cause the PC4500/4800 to start rescanning. This may be required due to a Deauthentication or Disassociation from the parent AP, or due to the link quality degrading to an unacceptable level.

Lose Sync Command				
Command (0x00)	Param0 (0x02)	Param1 (0x04)	Param2 (0x06)	
0x0003	0x0000	0x0000	0x0000	Lose Sync with the current cell – start rescanning.
Lose Sync Responses = Success				
Status (0x08)	Resp0 (0x0A)	Resp1 (0x0C)	Resp2 (0x0E)	Description
0x0003	0x0000	0x0000	0x0000	Successful Lose Sync command
Lose Sync Responses = Failure				
Status (0x08)	Resp0 (0x0A)	Resp1 (0x0C)	Resp2 (0x0E)	Reason
	Error Qualifier			
0x7F03	0x0000	0x0000	0x0000	Not Active – MAC must be enable first
0x7F03	0x0001	0x0000	0x0000	Illegal command – cannot be a root AP
0x7F03	0x0002	0x0000	0x0000	Illegal format (Command)

Soft Reset Command

This is equivalent to resetting the card.

Note, if this command completes successfully, it does not issue a command done.

Soft Reset Command				
Command (0x00)	Param0 (0x02)	Param1 (0x04)	Param2 (0x06)	
0x0004	0x0000	0x0000	0x0000	Restart (reboot) the card.
Soft Reset Responses = Failure				
Status (0x08)	Resp0 (0x0A)	Resp1 (0x0C)	Resp2 (0x0E)	Reason
	Error Qualifier			
0x7F04	0x0000	0x0000	0x0000	Illegal command – not supported
0x7F04	0x0001	0x0000	0x0000	Illegal format (Command)
0x7F03	0x0002	0x0000	0x0000	

Host Sleep Command

This is used by hosts to allow power save operation to achieve maximum power savings. After issuing this command, the host must not access the card until it is awakened by the host.

There are no responses to this command since the host must stop accessing the card after issuing the command.

Host Sleep Command				
Command (0x00)	Param0 (0x02)	Param1 (0x04)	Param2 (0x06)	
0x0085	0x0000	0x0000	0x0000	Host is going to sleep.

Read / Save Configuration Command

This command may be used by hosts to save and restore the configuration to/from nonvolatile storage. NOTE: the MAC must be disabled to issue this command.

The items saved in the configuration are: Current Configuration RID, SSID List RID, AP List RID and the Encapsulation RID

Read / Save Configuration Command				
Command (0x00)	Param0 (0x02)	Param1 (0x04)	Param2 (0x06)	
0x0008	0x0000	0x0000	0x0000	Restore the configuration from nonvolatile storage.
0x0108	0x0000	0x0000	0x0000	Save the configuration to nonvolatile storage

Magic Packet Command

This command is used by a host to tell the PC4500/4800 to start scanning for magic packets.

Magic Packet Command				
Command (0x00)	Param0 (0x02)	Param1 (0x04)	Param2 (0x06)	
0x0006	0x0000	0x0000	0x0000	Enter Magic Packet search mode

PSP Nodes Command

This command is issued by an AP host to indicate to the card that PSP nodes are associated. If PSP nodes are associated, then broadcast and multicast traffic will be deferred until the DTIM. If PSP nodes are not associated, then broadcast and multicast traffic will be sent when queued. (Refer to the section on AP specific information for more detail.)

Allocate Transmit Buffer Command

An allocate command cannot be issued until the previous allocate has completed (i.e. the alloc event has occurred and the TxAllocFid has been read).
The allocation size is the size of the payload portion of the packet to be transmitted. The payload size does not include the destination and source addresses.

Allocate Transmit Command				
Command (0x00)	Param0 (0x02)	Param1 (0x04)	Param2 (0x06)	
0x000A	Payload Size	0x0000	0x0000	Allocate Transmit Buffer
Allocate Transmit Responses = Success				
Status (0x08)	Resp0 (0x0A)	Resp1 (0x0C)	Resp2 (0x0E)	Description
0x000A	0x0000	0x0000	0x0000	Successful Allocate command
Allocate Transmit Responses = Failure				
Status (0x08)	Resp0 (0x0A)	Resp1 (0x0C)	Resp2 (0x0E)	Reason
	Error Qualifier			
0x7F0A	0x0002	0x0000	0x0000	Illegal format (Command)
0x7F0A	0x0005	0x0000	0x0000	Allocation size is too large
0x7F0A	0x0007	0x0000	0x0000	Alloc is already busy

Transmit Command

The TransmitFID parameter must be a previously allocated transmit buffer, which has been filled in with the appropriate control fields and the transmit packet (see FID description). Note that a successful completion of the transmit command (EvStat.Cmd) only indicates that the packet has been queued for transmission. When the transmission actually completes, the TxControl field of the TransmitFID controls the action taken. The TransmitFid may be returned to the free pool with no indication, a successful indication or a failure indication to the host. If the completion is indicated to the host, then the TransmitFID will be returned in the TxComplFID and the EvStat.Tx indicated if the transmit completed successfully or the EvStat.TxExc indicated if the transmission was not successful.

Note, well behaved firmware and drivers should never result in any of the transmit commands failing.

Transmit Command				
Command (0x00)	Param0 (0x02)	Param1 (0x04)	Param2 (0x06)	
0x000B	Transmit FID	0x0000	0x0000	Basic Transmit
Transmit Responses = Success				
Status (0x08)	Resp0 (0x0A)	Resp1 (0x0C)	Resp2 (0x0E)	Description
0x000B	0x0000	Transmit FID	0x0000	Successful transmit command. (Transmit may still fail however). The original Transmit FID value is returned to the host for matching purposes. However, the FID is under firmware control and cannot be reused by the host.
Transmit Responses = Failure				
Status (0x08)	Resp0 (0x0A)	Resp1 (0x0C)	Resp2 (0x0E)	Reason
	Error Qualifier	RID	RID Offset	
0x7F0B	0x0002	Transmit FID	0x0000	Illegal format (Command)
0x7F0B	0x0003	Transmit FID	0x0000	Invalid FID
0x7F0B	0x0005	Transmit FID	0x0000	FD.DataLen field is too long.

Deallocate Command

Used to deallocate a FID which is either a transmit buffer that is no longer to be used, or an allocated buffer no longer to be used.

Deallocate Command				
Command (0x00)	Param0 (0x02)	Param1 (0x04)	Param2 (0x06)	
0x000C	FID to release	0x0000	0x0000	Deallocate
Deallocate Responses = Success				
Status (0x08)	Resp0 (0x0A)	Resp1 (0x0C)	Resp2 (0x0E)	Description
0x000C	0x0000	0x0000	0x0000	Successful deallocation
Deallocate Responses = Failure				
Status (0x08)	Resp0 (0x0A)	Resp1 (0x0C)	Resp2 (0x0E)	Reason
	Error Qualifier			
0x7F0C	0x0002	0x0000	0x0000	Illegal format (Command)
0x7F0C	0x0003	0x0000	0x0000	Invalid FID

Access Resource Identifier (RID)

The Access Resource Identifier command is used to read and write configuration , status and statistics information. Each available resource identifier has a specific RID for accessing it. Note, only the most recently requested RID is available via the BAP registers.

The first word of all RIDs is the total length of the RID (in bytes) including this length field.

To read a RID	1.Use the Read RID command with the RID in the first parameter. 2.Read the RID using the BAP registers
To write a RID	1.Use the Read RID command with the RID in the first parameter. This ensures that the firmware BAP register access is correct. 2.Read the RID using the BAP registers. 3.Modify the appropriate fields using the BAP registers. 4.Write the RID using the Write RID command.

Access RID Command				
Command (0x00)	Param0 (0x02)	Param1 (0x04)	Param2 (0x06)	
0x0021	RID	0x0000	0x0000	Read RID
0x0121	RID	0x0000	0x0000	Write RID
Access RID Responses = Success				
Status (0x08)	Resp0 (0x0A)	Resp1 (0x0C)	Resp2 (0x0E)	Description
0x0021	0x0000	0x0000	0x0000	Successful access RID command
Access RID Responses = Failure				
Status (0x08)	Resp0 (0x0A)	Resp1 (0x0C)	Resp2 (0x0E)	Reason
	Error Qualifier			
0x7F21	0x0002	0x0000	0x0000	Illegal format (Command)
0x7F21	0x0004	0x0000	0x0000	Invalid RID
0x7F21	0x0006	0x0000	0x0000	Cannot write to this RID while MAC is enabled
0x7F21	0x000B	0x0000	0x0000	RID is write only
0x7F21	0x000C	0x0000	0x0000	RID is read only

Allocate Buffer Command

The Allocate Buffer command is used to pass variables used for transmitter testing (See Transmitter Test Command).

The allocated buffer is returned in PARAM0.

Allocate Buffer Command				
Command (0x00)	Param0 (0x02)	Param1 (0x04)	Param2 (0x06)	
0x0028	Buffer Size	0x0000	0x0000	Allocate Buffer
Allocate Buffer Responses = Success				
Status (0x08)	Resp0 (0x0A)	Resp1 (0x0C)	Resp2 (0x0E)	Description
0x0028	Buffer start	0x0000	0x0000	Successful allocate command
Allocate Buffer Responses = Failure				
Status (0x08)	Resp0 (0x0A)	Resp1 (0x0C)	Resp2 (0x0E)	Reason
	Error Qualifier			
0x7F28	0x0002	0x0000	0x0000	Illegal format (Command)
0x7F28	0x00??	0x0000	0x0000	No buffer space

Set PHY Register Command

The Set PHY register command takes:

Param0 = PHY register offset
Param1 = ClearBits
Param2 = SetBits

The PHY register will only be modified if ClearBits or SetBits are non-zero:
Phy = (Phy & (~ClearBits)) | SetBits;
In any event, the resultant (or existing if not modified) PHY value is returned in Resp0.

To read a PHY register, set ClearBits and SetBits to zero.

Note, the PC4500/4800 PHY specifies the register offsets in byte offsets, these have to be multiplied by two to get the word offset.

Special PHY Register Offsets map to local variables for controlling the transmitter tests as follows:

PC4500/4800 Direct Sequence PHY Register Description	
0x8000	PLCP word to precede all transmissions
0x8002	Frequency to be used for a transmitter test
0x8004	Transmit power (milliwatts)
0x8006	RSSI Threshold override

Transmitter Test Command

The transmitter test command accepts three optional parameters:

Param0 = command block
Param1 = frequency block
Param2 = pattern block

The command block contains a list of commands as follows:

0x0000	Ends the transmitter test
0x0001	Loop back to the beginning of the commands
0x0002	Start transmitting
0x0003	Stop transmitting
0x0004	Delay for N usec where N is the next word
0x0005	Delay for N Kusec where N is the next word
0x0006	Go to the next frequency in the frequency RID
0x0007	Start receive mode

Three blocks of memory must be pre-allocated using the Allocate Buffer command. These blocks are automatically freed when tests are completed normally. In the event that the tests are not completed normally, the blocks should be manually freed.

The frequency block contains a list of frequencies where 0 is 2400 MHz. The frequency block is only useful if there is a command block with the “Goto next frequency” command embedded in it. The pattern block contains a pattern of even length for transmission. A pattern block may be passed down regardless of whether there is a command block.

The following is sample code:

```
// note, these blocks of memory do not have the u16RidLen at the beginning
// used to transfer the command, frequency and pattern blocks to the PC4500/4800
```

```
static u16
data2rid(u16 *pDat, u16 lenDat, int *pStat)
{
   u16 rid;
   int status;

   if (pDat && lenDat) {
      rid = PC4500_allocbuf(lenDat);
      if (rid == 0) {
         *pStat = 1;
         return 0;
      }
      status = bap0_write(rid, 0, pDat, lenDat);
      if (status != 0) *pStat = status;
      return rid;
   }
   return 0;
}

// transmitter testing command
int PC4500_txtest(u16 *pCmd, u16 lenCmd,
                  u16 *pFreq, u16 lenFreq,
                  u16 *pPatt, u16 lenPatt)
{
   tds4500command cmd;
   tds4500response rsp;
   int status = 0;
   u16 ridCmd, ridFreq, ridPatt;
   ridCmd = 0;
   ridFreq = 0;
   ridPatt = 0;
   ridCmd = data2rid(pCmd, lenCmd, &status);
   ridFreq = data2rid(pFreq, lenFreq, &status);
   ridPatt = data2rid(pPatt, lenPatt, &status);
   if (status != 0) goto ERROR_EXIT;

   cmd.command = CMD_TXTEST;
   cmd.param[0] = ridCmd;
   cmd.param[1] = ridFreq;
   cmd.param[2] = ridPatt;
   status = PC4500_command(&cmd, &rsp);

   return status;

 ERROR_EXIT:
   /* free the allocated buffers */
   return status;
}
// for single channel receive mode testing
int PC4500_rxtest(void){
   tds4500command cmd;
   tds4500response rsp;
   memset(&cmd, 0, sizeof(cmd));
   cmd.command = CMD_TXTEST + 0x100;
   return PC4500_command(&cmd, &rsp);
}
```

Error Qualifier Values

ERROR QUALIFIER LIST	
Error Qualifier	**Description**
0x0000	No Qualifier (For Disable, it means already disabled!)
0x0001	Illegal command.
0x0002	Illegal format. Usually means some unsupported bits are set in the Command.
0x0003	Invalid FID.
0x0004	Invalid RID.
0x0005	Too Large
0x0006	MAC is not disabled.
0x0007	Alloc is still busy processing previous alloc
0x0008	Invalid Mode Field
0x0009	Tx is not allowed in monitor mode
0x000A	Loop test or memory test error
0x000B	Cannot read this RID.
0x000C	Cannot write to this RID.
0x000D	Tag not found.
0x0080	Config mode is invalid.
0x0081	Config hop interval is invalid.
0x0082	Config beacon interval is invalid.
0x0083	Config receive mode is invalid.
0x0084	Config MAC address is invalid.
0x0085	Config rates are invalid.
0x0086	Config ordering field is invalid.
0x0087	Config scan mode is invalid.
0x0088	Config authentication type is invalid.
0x0089	Config power save mode is invalid.
0x008A	Config radio type is invalid.
0x008B	Config diversity is invalid.
0x008C	Config SSID list is invalid.
0x008D	Config specified AP list is invalid.

Memory (FID/RID) Access

The PC4500/4800 provides two sets of 3 I/O registers for reading and writing receive and transmit packets and statistics and configuration information. It is suggested that one set be used in an interrupt context and the other in a non-interrupt context. These registers are collectively known as the "BAP" registers. (also see section "Basic FID Access" on page 7-14 for more information)

Transmit and receive packets are accessed using the Frame Identifiers (FID) that are passed to the host in the RxFid, TxComplFid and TxAllocFid registers.

Configuration, statistics and status information are accessed using a predefined Resource Identifier (RID) number.

Details on the contents of the FIDs and RIDs are available in later sections. This section explains the general mechanism for reading and writing FIDs and RIDs.

FID/RID are accessed using three I/O registers: Selector, Offset, Data.

The Selector register must be written first to choose the desired FID/RID. A selector value of zero is never used. The Offset register is written second to select the position within the FID/RID. Since only 16-bit accesses are supported, only even offsets are allowed -- the least significant bit must be zero. The third register is the Data register which allows reads and writes to the FID/RID. Successive reads and writes advance to the next sequential location within the FID/RID.

The following is sample code for reading/writing FID/RIDs:

```
bap0_setup(u16 rid, u16 offset)
{
   // this example uses SELECT0/OFFSET0/DATA0
   OUT4500(SELECT0, rid);
   OUT4500(OFFSET0, offset);
   while (1) {
      status = INPORT(OFFSET0);
      if (status & BAP_BUSY) {
        if (timeout) {
            // suggested timeout of 500 usec minimum
            push_interrupt_enable_state();      // save interrupt enable state
            disable_interrupts();               // disable interrupts
            OUT4500(SELECT0, rid);
            OUT4500(OFFSET0, offset);
            pop_interrupt_enable_state();       // restore interrupt enable state
            restart_timeout();
         }
         continue;
       }
      if (status & BAP_ERR) {
         // invalid rid or offset
         return ERROR;
      }
      if (status & BAP_DONE) {
         // success
         return SUCCESS;
      }
```

```
        // -- PC4500 missed it, try again
        OUT4500(SELECT0, rid);
        OUT4500(OFFSET0, offset);
    }
}

// requires call to bap0_setup() first
bap0_read(u16 *pu16Dst, int bytelen)
{
    bytelen = (bytelen + 1) & (~1);          // round up to even value
    while (bytelen > 0) {
        // each access to DATA0 will auto-increment to next word
        *pu16Dst++ = IN4500(DATA0);
        bytelen -= 2;
    }
    return SUCCESS;
}

// requires call to bap0_setup() first
bap0_write(u16 *pu16Src, int bytelen)
{
    bytelen = (bytelen + 1) & (~1);          // round up to even value
    while (bytelen > 0) {
        // each access to DATA0 will auto-increment to next word
        OUT4500(DATA0, *pu16Src++);
        bytelen -= 2;
    }
    return SUCCESS;
}
```

Note, only word writes are available, a read-modify-write is required to change a byte. In this case the offset register would have to be rewritten since it would auto-increment to the next address. In some cases, a read-modify-write is not necessary to write a single byte. For example, if writing a transmit packet of odd length, the last byte does not require a read-modify-write since the other byte will be ignored anyway.

Notes:

1) The Offset0/Offset1 registers are not updated by reads/writes to the Data0/Data1 registers.
2) Since the Offset0/Offset1 registers are not updated, an interrupt service routine cannot use the same set of registers for accessing FID/RIDs, since it cannot restore the original location prior to the interrupt.
3) A hardware errata prevents interleaved read/writes from operating correctly. The Selector/Offset must be reinitialized before changing from reads to write or vice-versa.

Memory items consist of:

FIDs - Frame Identifiers (transmit/receive packets or allocated memory)
RIDs - Resource Identifiers (configuration items)

Reading and Writing RIDs

Reading and writing RIDs requires use of commands in addition to the BAP registers.

Prior to reading or writing the RID, the Access RID command must be issued. Also after modifying a RID, the Access RID command must be used to commit the changes.

```
static tds4500command   cmd;          /* for issuing commands */
static tds4500response  rsp;          /* response from commands */

int PC4500_accessrid(u16 rid, u16 accmd)
{
   u16 status;

   memset(&cmd, 0, sizeof(cmd));
   cmd.command = accmd;
   cmd.param[0] = rid;

   status = PC4500_command(&cmd, &rsp);
   if (status != 0) return status;
   if ( (rsp.status & 0x7F00) != 0) {
      return (accmd << 8) + (rsp.response[0] & 0xFF);
   }
   return 0;
}


int PC4500_readrid(u16 rid, void *pBuf, int len)
{
   u16 status;

   if ( (status = PC4500_accessrid(rid, CMD_ACCESS)) != 0) return status;

   if (bap0_setup(rid, 0) != SUCCESS) return ERROR;

   // read the rid length field
   bap0_read(pBuf, 2);

   // length for remaining part of rid
   len = min(len, *(u16*)pBuf) - 2;

   // read remainder of the rid
   return bap0_read(((u16*)pBuf)+2, len);
}

int PC4500_writerid(u16 rid, const void *pBuf, int len)
{
   u16 status;

   // --- first access so that we can write the rid data
   if ( (status = PC4500_accessrid(rid, CMD_ACCESS)) != 0) return status;

   // --- now write the rid data
```

```
    if (bap0_setup(rid, 0) != SUCCESS) return ERROR;
    bap0_write(pBuf, len);

    // ---now commit the rid data
    return PC4500_accessrid(rid, 0x100|CMD_ACCESS);
}
```

Frame Info Descriptor

A Frame Information Descriptor (FID) is used to pass transmit and receive packets to and from the PC4500/4800. FIDs are passed to the host in the RxFid, TxAllocFid and TxComplFid registers. The FID is then accessed (read/written) using the BAP registers. The BAP registers consist of a Selector, Offset, Data registers. Details on using these registers is described in Memory (RID/FID) Access.

The FID consists of the following sections:

- Control header
- 802.11 header
- Gap for Protocol Encapsulation changes
- 802.3 header (may be disabled)
- Packet Payload.

The control header contains fields for selecting transmission options, and for reporting status of receive and transmit packets. When a transmit FID is allocated, the Control Header will be initialized to all zeros.

The 802.11 header allows for transmitting and receiving of 802.11 packets. Station drivers typically won't use the 802.11 header, since they only receive data packets. Instead, station drivers will use the 802.3 header and allow the PC4500/4800 to build the 802.11 transmission header. Similarly for receive packets, rather than having to interpret the 802.11 header, the station drivers will look at the 802.3 header that has been provided by the PC4500/4800. For receive packets, the 802.11 header is always available even if the 802.3 header is provided.

Access point hosts are required to interpret the 802.11 header since they must handle non-data packets (e.g. 802.11 management packets). Access point hosts will typically disable the 802.3 header for efficiency.

The PC4500/4800 provides for two style of hosts: ethernet and LLC. An ethernet style host is one which expects an ether-type field at the beginning of the packet payload. An LLC style host expects 802.2 (LLC) to be the first protocol in the payload. For an LLC host, the PC4500/4800 will transmit and receive packet payloads without modification.

Since 802.11 is an 802 network, the payloads require 802.2/LLC as the first protocol in the packet payload. Ethernet packets must be reencapsulated using RFC1042 or 802.1H, to be sent on 802.11. For LLC/802.2 hosts, the packet payloads are sent and received without any modifications. For ethernet hosts, the transmit and receive payloads are modified using the encapsulation rules that were configured.

The Protocol Gap field allows for adding and stripping these protocol modifications. The offsets shown for fields after the "gap" reflect a gap length of zero bytes. The offsets shown can be used to access the fields shown. The PC4500/4800 will adjust the access to the correct location. However, if reading through the gap, the host must skip past the appropriate number of locations.

Note, when a transmit FID is allocated, the Control header will be initialized to all zeros by the firmware, all other fields will contain uninitialized data.

FID with 802.3 Header – Station Mode

Station drivers will typically configure the PC4500/4800 to have 802.3 headers in the FID. This allows the driver to ignore the details of the 802.11 wireless network header and simply transfer ethernet or LLC packets.

For receive, this mode allows the host driver to efficiently read the most common fields with only one BAP setup. Typically the driver only needs to read the 802.3 Header and payload.

For transmit, the Control Header fields must be initialized, and then the 802.3 Header and payload written. If the transmit FID is being reused after completion, then the initialization of the Control Header need only be done the first time after the transmit FID is initially allocated. This allows for further efficiency, since the driver need only write the 802.3 Header and payload for each subsequent transmission.

The following table shows the fields in the receive and transmit FID with an 802.3 Header.

Offset	Receive FID	Transmit FID
Control Header		
+0x00	RxTime	SwSupport
+0x02		
+0x04	Status	Status
+0x06	PayloadLen	PayloadLen
+0x08	Rsv/Signal	TxControl
+0x0A	Rate/Freq	AID
+0x0C	RxAssocCnt/Rsv	TxRetries
+0x0E		TxAssocCnt/Rate
+0x10	PLCP	MaxLongRetry/MaxShortRetry
+0x12	PLCP…	
802.11 Header		
+0x14	FrameControl	FrameControl
+0x16	Duration	Duration
+0x18	Address1	Address1
+0x1A		
+0x1C		
+0x1E	Address2	Address2
+0x20		
+0x22		
+0x24	Address3	Address3
+0x26		
+0x28		
+0x2A	SeqCtl	SeqCtl
+0x2C	Address4	Address4
+0x2E		
+0x30		
Protocol GAP		
+0x32	GapLen	GapLen
----	Gap[ ]	Gap[ ]
802.3 Header -- may be disabled		
+0x34*	Status	Status
+0x36*	PayloadLen	PayloadLen
+0x38*	DestAddr	DestAddr
+0x3A*		
+0x3C*		
+0x3E*	SrcAddr	SrcAddr
+0x40*		
+0x42*		
Payload		
+0x44*	Payload[PayloadLen]	Payload[PayloadLen]

During transmission of ethernet payloads, the PC4500/4800 may overwrite the 802.3 Header Source Address (SrcAddr) with protocol encapsulation headers that are required to transmit the payload onto the wireless LAN.

Receiving an 802.3 Packet

The following is sample code for receiving a packet:

```
typedef struct {
   /* 802.3 packet header (802.3 packet format) */
   unsigned short  Status;
   unsigned short  PayloadLen;
   unsigned char   DstAddr[6];
   unsigned char   SrcAddr[6];
} tdsPktHdr;

struct {
   tdsPktHdr      header;
   char           payload[2312];
}
unsigned short Fid;

unsigned short receive_802_3_packet(void)
{
   if ( (IN4500(EVSTAT) & EV_RX) == 0) return NO_PACKET;

   Fid = IN4500(RXFID);

   if (bap0_setup(Fid, 0x34) != SUCCESS) return ERROR;

   // read the 802.3 header
   bap0_read(&rxpkt.header, sizeof(rxpkt.header));

   // read the packet payload
   if (rxpkt.header.PayloadLen) {
      bap0_read(&rxpkt.payload, rxpkt.header.PayloadLen);
   }

   // acknowledge reception
   OUT4500(EVACK, EV_RX);

   return SUCCESS;
}
```

Note, that this leaves the receive packet destination/source/payload in a suitable (contiguous) format for passing up to higher layers.

Transmitting an 802.3 Packet

The following is sample code for transmitting a packet:

```
typedef struct {                    /* transmit control header */
   unsigned long   SwSupport;      /* for use by host drivers */
   unsigned short  Status;
   unsigned short  PayloadLen;
   unsigned short  TxControl;
#define TXCTL_TXOK      (1<<1)         /* report if tx is ok */
#define TXCTL_TXEX      (1<<2)         /* report if tx fails */
#define TXCTL_802_3     (0<<3)         /* 802.3 packet */
#define TXCTL_802_11    (1<<3)         /* 802.11 mac packet */
#define TXCTL_ETHERNET  (0<<4)         /* payload has ethertype */
#define TXCTL_LLC       (1<<4)         /* payload is llc */
#define TXCTL_RELEASE   (0<<5)         /* release after completion */
#define TXCTL_NORELEASE (1<<5)         /* on completion returns to host */
} tdsTxCtlHdr;

tdsCommand Cmd;
tdsResponse Rsp;

unsigned short transmit_allocate(int lenPayload)
{
   Cmd.Command = CMD_ALLOCATETX;
   Cmd.Param0  = lenPayload;
   if (issuecommand(&cmd, &rsp) != SUCCESS) return 0;
   if ( (rsp.Status & 0xFF00) != 0) return 0;

   // wait for the allocate event/indication
   while ( (IN4500(EVSTAT) & EV_ALLOC) == 0) ;

   // get the allocated fid and acknowledge
   TxFid = IN4500(TXALLOCFID);
   OUT4500(EVACK, EV_ALLOC);
   return TxFid;
}

transmit_802_3_packet(char *pPacket, int len)
{
   unsigned short TxFid, TxControl, PayloadLen, EvStat, Status;

   if (len < 12) return ERROR;

   // packet is destination[6], source[6], payload[len-12]
   if ( (TxFid = transmit_allocate(len-12)) == 0) return ERROR;

   // write the Transmit control options
   TxControl = TXCTL_TXOK | TXCTL_TXEX | TXCTL_802_3
         | TXCTL_ETHERNET | TXCTL_RELEASE;
   if (bap0_setup(TxFid, 0x0008) != SUCCESS) return ERROR;
   bap0_write(&TxControl, sizeof(TxControl));

   // write the payload length and dst/src/payload
   if (bap0_setup(TxFid, 0x0036) != SUCCESS) return ERROR;
```

```
    PayloadLen = len - 12;
    bap0_write(&PayloadLen, sizeof(PayloadLen));
    bap0_write(pPacket, len);

    // issue the transmit command
    Cmd.Command = CMD_TRANSMIT;
    Cmd.Param0  = TxFid;
    if (issuecommand(&cmd, &rsp) != SUCCESS) return ERROR;
    if ( (rsp.Status & 0xFF00) != 0) return ERROR;

    // -- the following may be executed later in interrupt context

    // wait for the transmit to complete
    while ( ( (EvStat = IN4500(EVSTAT)) & (EV_TX|EV_TXEXC)) == 0) ;

    // get the allocated fid and acknowledge
    TxFid = IN4500(TXCOMPLFID);

    // optionally read the return status
    if (EvStat & EV_TXEXC) {
       if (bap0_setup(TxFid, 0x0034) != SUCCESS) return ERROR;
       bap0_read(&Status, sizeof(Status));
    }

    // acknowledge handling of the tx completion
    OUT4500(EVACK, EvStat & (EV_TX|EV_TXEXC));
}
```

Alternatively, the host may keep a fixed number of pre-allocated transmit FIDs which may be reclaimed when the TxComplFid is indicated to the host. The TxControl field should have the TXCTL_NORELEASE bit set in this case.

FID without 802.3 Header – Access Point Mode

On a host based Access Point, the card is typically configured to operate without 802.3 Headers. The Access Point reads all required information directly from the 802.11 header, so the 802.3 header is eliminated for efficiency.

The Receive 802.3 Header may be disabled by setting the PayloadType bit in the OperatingMode field of the general configuration.

The Transmit 802.3 Header is disabled by setting the PayloadType bit in the TxControl field in the Control header.

Protocol encapsulation changes are still required for ethernet hosts; hence the gap field is still present.

Receive 802.11 management and control packets will have a gap length of zero. Receive 802.11 data packets may have a gap length of 0, 2 or 8.

To allow for protocol encapsulation changes for transmission, a transmit FID gap of six bytes must be provided. (For FIDs with 802.3 Headers, recall that the 802.3 Header source address was reused for this purpose). The 6 byte gap is used even for management packets and for non-ethernet payloads.

When using 802.11 headers, the host should read/write through the gap. The host must not use the offset to jump to the payload location.

Offset	Receive FID	Transmit FID
Control Header		
+0x00	RxTime	SwSupport
+0x02		
+0x04	Status	Status
+0x06	PayloadLen	PayloadLen
+0x08	Rsv/Signal	TxControl
+0x0A	Rate/Freq	AID
+0x0C	RxAssocCnt/Rsv	TxRetries
+0x0E		Rate/TxAssocCnt
+0x10	PLCP	MaxLongRetry/MaxShortRetry
+0x12		
802.11 Header		
+0x14	FrameControl	FrameControl
+0x16	Duration	Duration
+0x18	Address1	Address1
+0x1A		
+0x1C		
+0x1E	Address2	Address2
+0x20		
+0x22		
+0x24	Address3	Address3
+0x26		
+0x28		
+0x2A	SeqCtl	SeqCtl
+0x2C	Address4	Address4
+0x2E		
+0x30		
Protocol GAP		
+0x32	GapLen	GapLen
----	Gap[ ]	Gap[ ]
In AP mode, the host must read through the gap		

To allow for protocol encapsulation changes for transmission, a transmit FID gap of six bytes must be provided. (For FIDs with 802.3 Headers, recall that the 802.3 Header source address was reused for this purpose).

Receiving an 802.11 Packet

The following is sample code for receiving a packet, assuming that the 802.3 header reception has been disabled:

```
typedef struct {          /* receive control header */
   unsigned long   RxTime;        /* time at start of 802.11 MAC */
   unsigned short  Status;        /* receive packet status */
#define RXS_CRCERR     (1<<1)     /* CRC32 error -- monitor mode only */
   unsigned short  PayloadLen;    /* length of payload */
   unsigned char   Reserved1;
   unsigned char   Signal;        /* rssi signal strength */
   unsigned char   Rate;          /* rate at which received 0x02=1Mbps */
   unsigned char   Frequency;     /* frequency received on 0=2400 MHz */
   unsigned char   RxAssocCnt;    /* AP ONLY */
   unsigned char   Reserved2[3];
   unsigned char   PlcpHeader[4];  /* Plcp Header */
} tdsRxCtlHdr;

typedef struct {          /* 802.11 mac header (802.11 packet format) */
   unsigned short  FrameControl;   /* PC4500 sets Retry, MoreFrag, Power, Order */
   /* Note, host sets FrameControl Retry for resubmit */
   unsigned short  Duration;   /* set by PC4500 */
   unsigned char   Addr1[6];
   unsigned char   Addr2[6];   /* ignored by PC4500 */
   unsigned char   Addr3[6];
   unsigned short  Sequence;   /* set by PC4500 if FrameControl.Retry is clear */
   unsigned char   Addr4[6];
} tdsMacHdr;

struct {
   tdsRxCtlHdr CtlHeader;  // receive control header
   tdsMacHdr      MacHeader;  // 802.11 mac header
   char           payload[2312];  // payload
}
unsigned short Fid;
unsigned short GapLen;
unsigned short Gap[8];

unsigned short receive_802_11_packet(void)
{
   if ( (IN4500(EVSTAT) & EV_RX) == 0) return NO_PACKET;

   Fid = IN4500(RXFID);

   // read the receive control header and 802.11 header
   if (bap0_setup(Fid, 0) != SUCCESS) return ERROR;
   bap0_read(&rxpkt.CtlHeader, sizeof(rxpkt.CtlHeader)+sizeof(rxPkt.MacHeader));

   // skip past the gap
   bap0_read(&GapLen, sizeof(GapLen));
   if (GapLen > 8) return ERROR;       // excessive gap
   bap0_read(&Gap, GapLen);

   // read the packet payload
```

```
    if (rxpkt.CtlHeader.PayloadLen) {
       bap0_read(&rxpkt.payload, rxpkt.CtlHeader.PayloadLen);
    }

    // acknowledge reception
    OUT4500(EVACK, EV_RX);

    return SUCCESS;
}
```

Transmitting an 802.11 Packet

When transmitting an 802.11 packet, the following fields must be filled in by the host:

Field	**Filled in by**
Frame Control:Type	Host
Frame Control:SubType	Host
Frame Control:FromDS	Host
Frame Control:ToDS	Host
Frame Control:MoreFrag	PC4500/4800
Frame Control:Retry	PC4500/4800 Host sets this for a resubmit
Frame Control:PowerMgt	PC4500/4800 -- not used on AP
Frame Control:MoreData	Host sets for unicast packets PC4500/4800 sets for broadcast / multicast packets
Frame Control:Wep	Host -- not supported yet
Frame Control:Order	PC4500/4800
Duration	PC4500/4800
Address1	Host
Address2	PC4500/4800
Address3	Host
SequenceControl	PC4500/4800 Host: sets this for a resubmits
Address4	Host
GapLength	--
Gap[6]	gap of six bytes is required
Payload	

Additionally, the host should ensure that the FrameControl bits MoreFrag, Retry, PowerMgt, and Order are zero for an initial submission of a packet.

If a transmission fails, the 802.11 packet may be resubmitted at a later time:

1.read and save the entire transmit FID
2.later resubmit the FID for transmission as follows
3.write the saved FID back into a PC4500/4800 FID
4.set the Frame Control Retry bit -- this is necessary to preserve the Sequence Control field
5.issue a normal transmit command for the FID

Alternatively, the host may resubmit the packet to be entirely retransmitted with a new sequence number. This is only possible if the MoreFrag bit is set, or the Retry bit is not set. This is done to avoid duplicate packets.

The following is sample code for transmitting an 802.11 packet:

```
typedef struct {          /* transmit control header */
   unsigned long   SwSupport;  /* for use by host */
   unsigned short  Status;     /* OUTPUT: zero is success */
#define TXS_RETRY   (1<<1)      /* max retries */
#define TXS_AGED    (1<<2)      /* max msdu lifetime exceeded */
#define TXS_CANCELLAID  (1<<3)      /* cancelled for failed AID */
#define TXS_MACDISABLED (1<<4)      /* mac was disabled by host */
#define TXS_LOSTASSOC   (1<<5)      /* lost association */
   unsigned short  PayloadLen;
   unsigned short  TxControl;
#define TXCTL_TXOK  (1<<1)      /* report if tx is ok */
#define TXCTL_TXEX  (1<<2)      /* report if tx fails */
#define TXCTL_802_3 (0<<3)      /* 802.3 packet */
#define TXCTL_802_11    (1<<3)      /* 802.11 mac packet */
#define TXCTL_ETHERNET  (0<<4)      /* payload has ethertype */
#define TXCTL_LLC   (1<<4)      /* payload is llc - (leave as is) */
#define TXCTL_NORELEASE (1<<5)      /* returns FID to host on tx complete */
#define TXCTL_USERTS    (1<<6)      /* forces use of RTS/CTS */
   unsigned short  AID;        /* AP ONLY - association ID */
   unsigned char   LongRetryUsed;  /* OUTPUT: number of retries used */
   unsigned char   ShortRetryUsed; /* OUTPUT: number of retries used */
   unsigned char   TxAssocCnt; /* AP ONLY - association count */
   unsigned char   TxBitRate;  /* AP must use, optional for client  to set data rate to use */
   unsigned char   MaxLongRetry;   /* AP ONLY - limit retries for packet */
   unsigned char   MaxShortRetry;  /* AP ONLY - limit retries for packet */
   unsigned char   Reserved[2];
} tdsTxCtlHdr;

unsigned short gapForTx802_11[] = { 6, 0, 0, 0};    /* six byte gap */

tdsCommand Cmd;
tdsResponse Rsp;

tdsTxCtlHdr TxCtlHdr;

transmit_802_11_packet(tdsMacHdr *p802Hdr, void *pPayload, int lenPayload)
{
   unsigned short TxFid, TxControl, PayloadLen, EvStat, Status;

   // allocate transmit packet
   if ( (TxFid = transmit_allocate(lenPayload)) == 0) return ERROR;

   // write the Transmit control options
   memset(TxCtlHdr, 0, sizeof(TxCtlHdr));
   TxCtlHdr.PayloadLen = lenPayload;
   TxCtlHdr.TxControl = TXCTL_TXOK | TXCTL_TXEX | TXCTL_802_11
         | TXCTL_ETHERNET | TXCTL_RELEASE;
   TxCtlHdr.AID = association ID for the destination;
   TxCtlHdr.TxBitRate = bit rate to use for the destination;
   TxCtlHdr.TxAssocCnt = current assoc count from last received
            (re)associate response packet;
   TxCtlHdr.MaxLongRetry = number of retries desired;
```

```
    TxCtlHDr.MaxShortRetry = numer of retries desired;

    if (bap0_setup(TxFid, 0x0000) != SUCCESS) return ERROR;
    bap0_write(&TxCtlHDr, sizeof(TxCtlHDr));

    // write the 802.11 header
    bap0_write(p802Hdr, sizeof(*p802Hdr));

    // write the gap
    bap0_write(&gapForTx802_11, sizeof(gapForTx802_11));

    // write the payload
    if (lenPayload != 0) bap0_write(pPayload, lenPayload);

    // issue the transmit command
    Cmd.Command = CMD_TRANSMIT;
    Cmd.Param0  = TxFid;
    if (issuecommand(&cmd, &rsp) != SUCCESS) return ERROR;
    if ( (rsp.Status & 0xFF00) != 0) return ERROR;

    // -- the following may be executed later in interrupt context

    // wait for the transmit to complete
    while ( ( (EvStat = IN4500(EVSTAT)) & (EV_TX|EV_TXEXC)) == 0) ;

    // get the allocated fid and acknowledge
    TxFid = IN4500(TXCOMPLFID);

    // optionally read the return status
    if (EvStat & EV_TXEXC) {
       if (bap0_setup(TxFid, 0x0004) != SUCCESS) return ERROR;
       bap0_read(&Status, sizeof(Status));
    }

    // acknowledge handling of the tx completion
    OUT4500(EVACK, EvStat & (EV_TX|EV_TXEXC));
}
```

FID Field Details

Note, all fields are stored least significant byte first.

Offset	Name	Type	Description
Receive Control Header			
+0x0000	RxTime	u32	Receive time (beginning of packet) in usec relative to the TSF of the current cell.
+0x0004	Status	u16	The status field for receive FIDs will return as zero. 0x0002 – CRC32 error (RF Monitor Mode Only)
+0x0006	PayloadLen	u16	The length of the payload. (Does not include the Mac Header or 802.3 Header).
+0x0008	Reserved	u8	
+0x0009	Signal	u8	The RSSI value while receiving the packet.
+0x000A	Rate	u8	The rate at which the packet was received. Value in multiples of 500 Kbps. (0x02=1Mbps, 0x04=2Mbps).
+0x000B	Frequency	u8	The frequency on which the packet was received. (0=2400 MHz, 1=2401 MHz, ...).
+0x000C	RxAssociationCount	u8	The receive association count (only used for AP mode).
+0x000D	Reserved	u8[3]	
+0x0010	PlcpHeader	u8[4]	This is the received 802.11 PLCP Header.
Transmit Control Header			
+0x0000	Software support field	u32	This field is for use by the host and is not referenced by the firmware.
+0x0004	Status	u16	Transmit FID status field is bit-mapped as follows: 0x0002 Too many retries 0x0004 Transmit lifetime exceeded 0x0008 Cancelled due to AID failure (AP only) 0x0010 Cancelled due to MAC being disabled 0x0020 Cancelled due to association lost (AP only)
+0x0006	PayloadLen	u16	The length of the payload. (Does not include the Mac Header or 802.3 Header).
+0x0008	TxControl	u16	Transmit control is bit encoded as follows: Bit – Field – Description 0 – Reserved 1 – TxOk – Generate EvStat.Tx event and TxComplFid if transmit completes ok 2 – TxEx – Generate EvStat.TxEx event and TxComplFid if transmit fails 3 – Type – 0 Ethernet (802.3)packet; 1 802.11 packet 4 – PayloadType – 0 Ethernet payload -- ie. has an ethertype; 1 LLC payload -- no ethertype 5 – NoRelease – FID is returned to host for reuse when transmit completes. Note, the FID is returned in the **TxComplFid** (not TxAlloc). Note, if this bit is set, the TxOk and TxEx must also be set. 6 – NoRetries – Used to indicate that the packet should be transmitted without retries.

			7 ClearAID — Used only by an AP!!! Used to clear the failed transmit state of an association ID. Once a packet fails to a particular AID, all packets to that AID will be returned to the AP (without any attempts), until the ClearAID bit is set. This mechanism allows flushing of queued packets to maintain the correct packet order. 8 Strict Order — Indicates a strictly ordered multicast packet. The packet will not be queued for transmission after the DTIM, but rather will be sent with normal priority. See 802.11 specification for details on strictly ordered. 9 Use RTS — Forces the use of RTS/CTS for transmission of this packet. This is useful for resubmission of previously failed packets to start transmission with RTS/CTS enabled.
+0x000A	AID	u16	Association ID. Used only by an AP!!!.
+0x000C	TxRetry	u16	**Output** Retry Status for the FID from the PC4500/4800. This contains the number of retries required to transmit the packet. The upper byte is the short retry count. The lower byte is the long retry count. (Number of retries reported will be limited to 255 for both short/long). This field will return a zero if no retries were required.
+0x000E	TxAssociationCount	u8	Used only by an AP Repeater!!!. Transmissions will be returned to the host with a status Of Cancelled due to loss of association, if this field does not match the current association count. This is done to allow for retargeting of packets when reassociation to a new parent access point occurs.
+0x000F	Transmit Bit Rate	u8	Must be used by an AP!!!. Selects the desired transmission rate. (Value in multiples of 500 Kbps. 0x02=1Mbps, 0x04=2Mbps). A value of 0 for a client means the client will choose the best rate for delivery.
+0x0010	MaxLongRetries	u8	Used only on AP. Maximum long retry count.
+0x0011	MaxShortRetries	u8	Used only on AP. Maximum short retry count.
+0x0012	Reserved	u8[2]	
802.11 Header			For transmit, the following section must be filled in if the TxControl.Type is 802.11.
+0x0014	FrameControl	u16	Frame Control Word as defined in 802.11.
+0x0016	Duration	u16	The duration field as defined in 802.11. This field will always be overridden by the PC4500/4800
+0x0018	Addr1	u8[6]	The address1 field as defined in 802.11.
+0x001E	Addr2	u8[6]	The address2 field as defined in 802.11.
+0x0024	Addr3	u8[6]	The address3 field as defined in 802.11.
+0x002A	Sequence	u16	The sequence control field as defined in 802.11. This field will always be overridden by the PC4500/4800.
+0x002C	Addr4	u8[6]	The address4 field as defined in 802.11. Note, this field will only be used if both the FromDS and ToDS bits are set in the FrameControl.
+0x0032	GapLen	u16	The length of the Gap field in bytes. On receive, the GapLen will be 0, 2 or 8. For transmit FIDs with a 802.3 header, the gap len is 0. For transmit FIDs without 802.3 header, the gap len is 6.

+0x0034	Gap[ ]	u8[GapLen]	The gap field allows for changes in protocol encapsulation by the firmware.
802.3 Header			For transmit, the following section must be filled in if the TxControl.Type is 802.3.
+0x0034**	Status	u16	This is a duplicate of the status field from the control header.
+0x0036**	PayloadLen	u16	This is a duplicate of the payload length from the control header.
+0x0038**	DstAddr	u8[6]	Destination address.
+0x003E**	SrcAddr	u8[6]	Source address. Note, the source address will always be overridden to use the address assigned during configuration. This field need not be filled in. This field may be overwritten to add protocol encapsulation headers (802.1H/RFC1042) to the packet payload.
+0x0044**	Payload	u8[PayloadLen]	The packet payload.

Note, the offsets shown for the 802.3 Header and Packet Payload fields may be used to jump to the specified field when executing a BAP setup but only for an 802.3 packet. When transmitting/receiving an 802.11 style packet, the host should read/write through the gap after reading/writing the 802.11 header.

Resource Identifiers

Resource Identifiers (RIDs) are used to access configuration, status and statistics from the PC4500/4800.

Selector	Access	Description	Notes
Configuration (these RIDs cannot be written while the MAC is enabled)			
0xFF10	See notes	General Configuration	Many configuration items.
0xFF11	See notes	Valid SSID list	List of SSIDs which the station may associate to.
0xFF12	See notes	Valid AP list	List of APs which the station may associate to.
0xFF13	See notes	Driver name	The name and version of the driver (for debugging)
0xFF14	See notes	Ethernet Protocol Encapsulation	Rules for encapsulating ethernet payloads onto 802.11.
0xFF15	See notes	WEP Key volatile	Key used for encryption
0xFF16	See notes	WEP Key non volatile	Key used for encryption
0xFF17	See notes	Modulation	Set the default modulation to CCK or MBOK (4800 only)
0xFF20	Read only	Actual Configuration	This has the same format as the General Configuration. However, all fields are filled in with the actual values used.
Reporting			
0xFF00	Read Only	Capabilities	PC4500/4800 Information
0xFF01	Read Only	AP Info	Access Point Information
0xFF02	Read Only	Radio Info	Radio Information -- note radio specific
0xFF50	Read Only	Status	PC4500/4800 Current Status Information
Statistics			
0xFF60	Read Only	16-bit Statistics	Cumulative 16-bit Statistics
0xFF61	Read Only	16-bit Statistics	Delta 16-bit Statistics (since last clear)
0xFF62	Read Only / Clear	16-bit Statistics	Delta 16-bit Statistics and Clear
0xFF68	Read Only	32-bit Statistics	Cumulative 32-bit Statistics
0xFF69	Read Only	32-bit Statistics	Delta 32-bit Statistics (since last clear)
0xFF6A	Read Only / Clear	32-bit Statistics	Delta 32-bit Statistics and Clear

Once the MAC has been enabled, the configuration RIDs cannot be written (it can be read). The MAC must be disabled before the configuration can be modified.

Using the 0xFF60 and 0xFF68 RIDs will return the cumulative statistics which always accumulate and will never clear (although they may roll-over).
Using the 0xFF61 and 0xFF69 RIDs will return the delta statistics which are the statistics that have accumulated since the last clear.
Using the 0xFF62 and 0xFF6A Rids will return the delta statistics which are the statistics that have accumulated since the last clear . Additionally, all delta statistics will be zeroed.

General Configuration Parameters

The following describes the general configuration block.

Offset	Name	Type	Driver value	Factory default	Description
+0x0000	u16RidLen	u16	read-only	read-only	Length of this RID including the length field
+0x0002	OperatingMode	u16	**Must set**	1	Select mode of operation: 0 = IBSS/Adhoc 1 = Infrastructure - Station 2 = Access Point 3 = Access Point – Repeater The following bit-encoded fields are available for all the modes: Bit 8 (0x0100) PayloadType If this bit is set, packet payloads will be received without modification. If this bit is clear, the receive LLC payload will be converted to an ethernet style payload (starting with ethertype) using the encapsulation rules. See Encapsulation Rid for details. Bit 9 (0x0200) Aironet Extensions Enabled. Enables transmission of the Aironet Information Element. Bit10 (0x0400) Enable Access Point Extensions. See Access Point Interface for details.
+0x0004	ReceiveMode	u16	0	0	Receive Mode: -- separate this out some more... 0 = BC/MC/ADDR 1 = BC/ADDR 2 = ADDR 3 = 802.11 Monitor -- any destination -- only for the current BSSID 4 = 802.11 Monitor -- any destination – accept packets for any BSSID 5 = LAN Monitor -- any destination -- only for current BSSID The following bit-encoded fields are available for all the modes: bit 8 (0x0100) Disable 802.3 Header Disables the 802.3 header in receive Fids for those hosts that only read the 802.11 header.
+0x0006	FragmentThreshold	u16	0 default	700	Fragmentation Threshold (see 802.11). 0 selects factory default. Value will be rounded up to next even integer of at least 256 octets.
+0x0008	RTSThreshold	u16	0 default	300	RTS Threshold (see 802.11). 0 selects factory default. To have all packet exchanges preceeded by RTS/CTS select a non-zero value less than 24 (802.11 header length).
+0x000A	StationMacId	Addr	0 default	Factory	If zero, address assigned at the factory will be used.
+0x0010	Supported Rates	u8[8]	0 default	0x02, 0x04	Selects the rates which are to be supported. 0's selects the factory defaults. The entry of this field follows the 802.11coding for the rates information element. Each byte represents a rate in units of 500 Kbps (1 Mbps = 0x02). A "basic" rate is represented by setting the most significant bit. A "basic" 1 Mbps rate would be 0x82. The table ends with 0x00. 0x0002 = 1 Mbps 0x0004 = 2 Mbps
+0x0018	ShortRetryLimit	u16	0 default	16	Short Retry Limit (see 802.11). 0 selects factory default.

+0x001A	LongRetryLimit	u16	0 default	16	Long Retry Limit (see 802.11). 0 selects factory default.
+0x001C	TxMSDULifetime	u16 (kus)	0 default	5000	0 will select the factory default [~5 sec].
+0x001E	RxMSDULifetime	u16 (kus)	0 default	10000	0 will select the factory default [~10 sec]
+0x0020	Stationary	u16	0	0	If set, indicates that the radio is stationary. This information may be used to modify roaming algorithms.
+0x0022	Ordering	u16	0	0	If set, strictly ordered multicast/broadcasts are used.
+0x0024	Device Type	u16	0	0	The default for the PC4500 is 0x0065. The default for the PC4800 is 0x006D.
+0x0026	Reserved	u8[10]	0's	0	reserved for future use
SCANNING/ASSOCIATING					
+0x0030	ScanMode	u16	0	0	Default is active scanning 0 = Active -- default 1 = Passive 2 = Aironet Active Scanning
+0x0032	ProbeDelay	u16 (kus)	0 default	3	Time to wait after switching to a channel for clear channel assessment. 0 selects factory default. Use 0xFFFF to select 0 kus.
+0x0034	ProbeEnergyTimeout	u16 (kus)	0 default	3	Time to wait for energy after an active probe. 0 selects factory default.
+0x0036	ProbeResponseTimeout	u16 (kus)	0 default	20	Time to wait for a probe response after energy detected. 0 selects factory default.
+0x0038	BeaconListenTimeout	u16 (kus)	0 default	40	Time to listen for a beacon on each channel. 0 selects factory default.
+0x003A	IbssJoinNetTimeout	u16 (kus)	0 default	10000	IBSS: Time to scan for an IBSS before forming a network. 0 selects the factory default. 0xFFFF indicates scan until found. 1 is good enough for starting without scanning, since chances of finding a network in 1 kusec are virtually nil.
+0x003C	AuthenticationTimeout	u16 (kus)	0 default	2000	Time limit after which an authentication sequence will be terminated/restarted. 0 selects factory default.
+0x003E	AuthenticationType	u16	0 default	1 (open)	Selects the desired authentication and privacy methods. 0x01 = Open 0x02 = Shared-Key 0x04 = Exclude Unencrypted Note, 0 (zero) selects no authentication and no exclusion of unencrypted frames.
+0x0040	AssociationTimeout	u16 (kus)	0 default	2000	ESS: Time limit after which an association sequence will be terminated. 0 selects factory default.
+0x042	SpecifiedAPtimeout	u16 (kus)	0 default	10000	0 selects the factory default [~10 sec]. If non-zero, associate to any AP after the timeout interval has passed. A value of 0xFFFF indicates that the unit should never Associate to an AP not in the list. Note, only applies in ESS mode and if a "valid AP-list" has been passed down.
+0x0044	OfflineScanInterval	u16 (kus)	0	0	0 disables offline scanning. The time period between offline scans.
+0x0046	OfflineScanDuration	u16 (kus)	0	0	0 disables offline scanning. The duration of an offline scan.
+0x0048	LinkLossDelay	u16 (kus)	0	0	Time to delay before reporting a loss of association event. This may be required to prevent the driver from being overloaded with loss/reassociation events when the radio enters a fringe area. A loss of association will only be reported if the condition continues for longer than the LinkLossDelay time. A reassociation will always be reported (immediately) if the access point has changed. If the radio reassociates to the same AP before the LinkLossDelay time period has expired, no reassociate message will be posted up to the host.

+0x004A	MaxBeaconLostTime	u16 (kus)	0 default	500	If no beacons are received for this time period, the unit will begin rescanning. 0 selects factory default. Internally this will be converted to the number of consecutively missed beacons that may occur before rescanning. A minimum of eight consecutive missed beacons starts rescanning.
+0x004C	RefreshInterval	u16 (kus)	0 default -1 disables	10000	At the specified interval, the station will send a refresh (null packet) to the AP. 0 selects factory default. 0xFFFF is used to turn off refresh. Refresh packets serve two purposes: verify the AP is still present, and indicate to the AP that the station is still associated. The AP will send a disassociate packet in response to the refresh packet if the AP has staled out the association.
POWER SAVE OPERATION					
+0x0050	PowerSaveMode	u16	0	0	Note, for IBSS there is only one PSP mode and it is only enabled if the ATIMwindow is non-zero. 0 = CAM 1 = PSP 2 = PSP-CAM [FASTPSP]
+0x0052	SleepForDTIMs	u16	0	0	If non-zero, the station may sleep through DTIMs; this may result in the station missing broadcast/multicast traffic. If zero, the station is required to waken for each DTIM. Note, this parameter only applies to stations in infrastructure/ESS mode
+0x0054	ListenInterval	u16 (kus)	0 default	200 kus	Maximum time to awaken for TIMs. 0 selects factory default. (2 minute maximum) Note, this parameter only applies to stations in infrastructure/ESS mode
+0x0056	FastListenInterval	u16 (kus)	0 default	100 kus	The listen interval to be used immediately after receiving packets. 0 selects factory default. Note, this parameter only applies to stations in infrastructure/ESS mode
+0x0058	ListenDecay	u16	0 default	2	Number of times to use the current listen interval before doubling it. 0 selects factory default. Note, this parameter only applies to stations in infrastructure/ESS mode
+0x005A	FastListenDelay	u16 (kus)	0 default	200 kus	Time interval to delay before going to fast listen interval after transmitting. 0 selects factory default. Note, this parameter only applies to stations in infrastructure/ESS mode
+0x005C	Reserved	u8[4]			
ADHOC (or AP) OPERATION					
+0x0060	BeaconPeriod	u16 (kus)	0 default	100	0 selects the factory default of [~100 ms].
+0x0062	AtimDuration	u16 (kus)	0 default	5 kus	The time period reserved for ATIMs immediately after the beacon. 0xFFFF will disable the ATIM window; power save mode will not operate. This parameter only applies to adhoc/IBSS.
+0x0064	Reserved	u16	0	0	Reserved for future use
+0x0066	DSChannel	u16	0 default	1	The desired operating channel. ()refer to 802.11) For North America, a Channel of 0 is 2412 MHz.
+0x0068	Reserved	u16	0	0	Reserved for future use
+0x006A	DTIM Period	u16	0 default	1	Selects how often a beacon is a DTIM for APs
+0x006C	Reserved	u8[4]	0's	0's	Reserved for future use
RADIO OPERATION					
+0x0070	RadioType	u16	0 default	0	Selects the radio operational mode. By default, this will select the 802.11 radio that corresponds to the hardware. Check the capabilities to see if this may be changed. 0 = 802.11 FH Radio (Default) 1 = 802.11 DS Radio 2 = LM2000 (Legacy) DS Radio

+0x0072	Diversity	u16	0 default	0x0303	This field is bit-mapped to select the operational antennas. The lower byte selects active receive antennas. The upper byte selects active transmit antennas. 0 = Diversity as programmed at the factory 1 = Antenna 1 only 2 = Antenna 2 only 3 = Antennas 1 and 2 are active
+0x0074	TransmitPower	u16 (mw)	0 default	250 or 100	0 selects the default (maximum power allowed for the country/regulatory domain). 0 = As programmed at the factory 1-50 = 50 mw output (if not available, first available lower power level is used) 51-100 = 100 mw output (if not available, first available lower power level is used) 101-250 = 250 mw output
+0x0076		u16	0 default	0	No longer valid
+0x0078	Modulation Type	u16	0 default	1	4800 only: Selects between MOK and CCK modulation. 1 = MOK 2 = CCK
+0x007A	Reserved	u8[8]	0's	0's	reserved for future radio specific parameters
AIRONET EXTENSIONS					
+0x0080	NodeName	u8[16]	0	0	Station name.
+0x0090	ARLThreshold	u16	0 default	0xFFFF	0 selects the factory defaults. (which for now is "OFF").
+0x0092	ARLDecay	u16	0 default	0xFFFF	0 selects the factory defaults. (which for now is "OFF").
+0x0094	ARLDelay	u16 (kus)	0 default	0xFFFF	0 selects the factory defaults. (which for now is "OFF").
+0x0096	Unused				
+0x0097	Unused				
+0x0098	MagicPacketAction	u8	0	0	0 selects no action to be taken on a magic packet and disables magic packet mode.
+0x0099	MagicPacketControl	u8	0	0	0 will disable the magic packet mode command (it will return invalid command).

Additional information on the configuration elements follows:

The PC4500/4800 may be configured to operate as a station (either within an ESS while associated to an AP or within an IBSS/adhoc network), to operate as part of an access point or repeater or to operate as a wireless LAN monitor. It is important to recognize that when operating in infrastructure mode, some parameters (such as hop sequences) are adopted from the access point.

The correct order to configure a PC4500/4800 is to read the current configuration, make modifications as necessary, disable the MAC, write the new configuration, re-enable the MAC.

Station Operation

In station mode (ESS or IBSS), only data packets will be transferred to and from the host. No 802.11 native packets (management and control) will be passed to the host. All 802.11 management and control packets are handled by the PC4500/4800. It is recommended that the 802.3 header be used to pass transmit and receive packets between the host and PC4500/4800. This is easier for the host and more efficient. The host doesn't need to interpret or create an 802.11 header for each packet. The host can transfer the packets as is -- destination and source addresses, and packet payload.

Station ESS Operation

In station mode, the PC4500/4800 will scan for an appropriate wireless network, and then associate to an access point. The PC4500/4800 will accept transmit packets at any time, even if

the PC4500/4800 is not currently associated to an access point. The packet will be transmitted when the PC4500/4800 does finally associate, or will be removed from the queue when the transmit lifetime expires for the packet whichever occurs first.

Station IBSS Operation

In IBSS mode, also referred to as adhoc mode, the PC4500/4800 will scan for an appropriate wireless network and then join that network by synchronizing to the hopping pattern (if enabled). If the PC4500/4800 fails to find an appropriate network within the desired interval, it will form a new network; it may be the first station to be turned on. In IBSS mode, all stations must be in range of each other station to communicate with them.

Access Point Operation

In access point mode, the host must receive and transmit management and data 802.11 packets. The host must also receive PS-POLL control packets. In this mode, the PC4500/4800 provides only part of the access point operation; the host must provide a some of the functionality. Access point operation is supported for both "root" access points and "repeater" access points. (A repeater is an access point that registers to another access point to extend the range of a cell.)

In access point mode, it is recommended that the 802.3 headers be disabled for efficiency. The access point must deal with 802.11 headers in any event.

Wireless LAN Monitor Operation

In the monitor mode, the PC4500/4800 will scan for the appropriate network and then synchronize to the hopping pattern (if enabled) and begin passing receive packets to the host. In monitor mode, the host is not allowed to transmit.

Three modes of operation are supported for the wireless monitor:

wireless monitor for a single BSS
wireless monitor for any BSS
LAN monitor

The wireless monitor mode is intended for monitoring the wireless traffic as is. 802.11 management, control and data packets will all be passed to the host. 802.3 headers can be disabled. In wireless monitor mode, the packet payload is always passed through “as is” regardless of the setting of the payload type field. The single BSS mode limits received packets to the current BSSID.

The LAN monitor mode will only pass up data packets. In this case 802.3 headers should be used. This mode is useful for looking for upper layer protocol problems on the wireless LAN.

Packet Header Type

The transmit and receive packets always have an 802.11 wireless LAN protocol header. An optional 802.3 header, with destination and source addresses, is also available. The 802.3 header can be disabled for receive packets by setting the HeaderType bit the ReceiveMode field of the General Configuration. The 802.3 header can be disabled for transmit packets by setting the HeaderType bit in the TxControl field of the transmit FID.

For station mode, the preferred operation is with 802.3 headers, since the station need only receive and transmit data packets. For access point mode, 802.11 headers must be used. Disabling the 802.3 header allows for more efficient reading and writing of 802.11 packets since the host does not have to read or write the 802.3 header bytes.

Payload types

The PC4500/4800 may be configured to transmit and receive packet payloads as is (LLC host) or as ethernet payloads. Ethernet payloads have an ethertype field as the first word of the payload. These payloads must be modified for transmission on the 802.11 network. Details of this are discussed under encapsulation.

The receive payload type is selected with the PayloadType bit of the OperatingMode field of the General Configuration. The transmit payload type is selected with the PayloadType bit of the TxControl field of the transmit FID.

Aironet Extensions

Aironet provides some proprietary extensions to 802.11. The primary extension is the Aironet Information Element that is transmitted in some management packets. This element contains additional information about the access point and stations.

The Aironet extensions are enabled automatically by stations if the access point is transmitting the Aironet Information Element. The Aironet extensions are enabled on an access point by setting the AironetExtensions bit in the OperatingMode field of the General Configuration.

Access Point Interface

The access point interface is described in more detail in a separate section.

Receive Mode

This field should only be used in station mode. The receive mode field may be used to limit the traffic passed to the host by the station. It may be used to discard broadcast and multicast traffic rather than pass them to the host. This field is mainly intended for slower serial (RS232) hosts where broadcast and multicast traffic could significantly delay real traffic destined to the host.

Device Type

The device type field should be left as a zero for normal operation. The device type field is provided to allow an override of the default device type in the Aironet information element (Aironet extension) to distinguish different products within the Aironet product line.

802.11 Configuration Parameters

Several 802.11 configuration parameters are provided. Please see the 802.11 specification for additional details.

Fragment Threshold

Decreasing/shortening the fragment threshold will cause long packets to be sent as many shorter fragments. The shorter packets will be reassembled at the receiving station. Shortening the fragments increases the chance of delivering the packet to the destination since shorter packets have less chance of a bit error. However shortening the fragments also increases the bandwidth consumed. Under ideal conditions, the throughput will decrease with shorter fragments.

RTS Threshold

This value defines the packet length above which all larger packets will be transferred using the RTS/CTS reservation protocol.

Station Mac Address

Used to override the factory assigned address.

Power Save Operation

Power save operation is only supported in station mode. Access points are not allowed to operate in power save mode.

Power Save Support by the Host

For maximum power savings some cooperation is required from the host. Maximum savings can be achieved if the PC4500/4800 is aware that the host will not access the PC4500/4800. Mechanisms are provided to allow the host to go to sleep and the PC4500/4800 to wake the hose as well as for the host to awaken the PC4500/4800. Details are provided in PSP Cooperation by host.

Power Save Operation for ESS Station

Power Save Operation for IBSS Station

Modulation

This rid sets the default modulation of the card. The value will reside in non-volatile storage, on the card. This value can be overwritten by the configuration. This rid only applies to the 4800.

Offset	Name	Type	Factory default	Description
+0x0000	u16RidLen	u16	read-only 0x04	Length of this RID including the length field
+0x0002	u16Modulation	u16	0	CCK = 1, MBOK = 2

WEP Key Volatile

Reading rid 0xFF15 returns the first key. Reading rid 0xFF15 does not return the key. It returns the u16KeyLen. Non-zero values in these fields can be used to determine if a key has been set. Writing this rid sets the key in volatile storage.

The only valid length for u16KeyLen is 5 or 0.
The address {1, 0, 0, 0, 0, 0} is used to denote the default key. This is also the only valid address.
U16KeyIndex is always 0.

Offset	Name	Type	Factory default	Description
+0x0000	u16RidLen	u16	read-only 0x1C	Length of this RID including the length field
+0x0002	u16KeyIndex	u16	0	Index to list of keys
+0x0004	u8Addr[6]	u8[6]	1,0,0,0,0,0	The address associated with the key.
+0x000A	u16KeyLen	u16	0	
+0x000C	u8Key[16]	u8[16]	0's	The key.

WEP Key Non-volatile

Reading rid 0xFF16 returns the next key. Reading rid 0xFF16 does not return the key. It returns the u16KeyLen. Non-zero values in these fields can be used to determine if a key has been set. Writing this rid sets the key in non-volatile storage.

The only valid length for u16KeyLen is 5 or 0.
The address {1, 0, 0, 0, 0, 0} is used to denote the default key. This is also the only valid address.
U16KeyIndex is always 0.

Offset	Name	Type	Factory default	Description
+0x0000	u16RidLen	u16	read-only 0x1C	Length of this RID including the length field
+0x0002	u16KeyIndex	u16	0	Index to list of keys
+0x0004	u8Addr[6]	u8[6]	1,0,0,0,0,0	The address associated with the key.
+0x000A	u16KeyLen	u16	0	
+0x000C	u8Key[16]	u8[16]	0's	The key.

Valid SSID List

The SSID RID contains up to three SSIDs that may be matched. This allows a user to configure the unit for multiple sites.
If SSIDlen1 is zero, then any SSID will be accepted as valid.
Use SSIDlen2 equal zero or SSIDlen3 equal zero to indicate the end of SSID list.

Note, the driver must set these values as desired.

Offset	Name	Type	Factory default	Description
+0x0000	u16RidLen	u16	read-only 0x68	Length of this RID including the length field
+0x0002	SSIDlen1	u16	7	The length of the SSID1 byte string.
+0x0004	SSID1[32]	u8[32]	"tsunami"	The identifier uniquely identifying the wireless system.
+0x0024	SSIDlen2	u16	0	The length of the SSID2 byte string.
+0x0026	SSID2[32]	u8[32]	0's	The identifier uniquely identifying the wireless system.
+0x0046	SSIDlen3	u16	0	The length of the SSID3 byte string.
+0x0048	SSID3[32]	u8[32]	0's	The identifier uniquely identifying the wireless system.

Valid AP List

The AP list contains up to four specified APs that may be matched. This allows a user to limit a unit to a subset of APs. Multicast addresses are not allowed. A zero address ends the list.
Note, all addresses should be null (all zero's) to disable the specified AP feature.

Offset	Name	Type	Factory default	Description
+0x0000	u16RidLen	u16	read-only 0x20	Length of this RID including the length field
+0x0002	SpecifiedAP1	u8[6]	0	Specifies the MAC address of an access point to attempt to associate to first, before looking for other Access Points
+0x0008	SpecifiedAP2	u8[6]	0	Allows for a secondary AP to associate to if the radio cannot associate to the primary AP.
+0x000E	SpecifiedAP3	u8[6]	0	Allows for a third option when specifying a list of APs.
+0x0014	SpecifiedAP4	u8[6]	0	Allows for a fourth option when specifying a list of APs.

Driver Name

The driver name and version may be passed down from the host for debug support. This RID is treated as a read-only value when the MAC is enabled.

Offset	Name	Type	Factory default	Description
+0x0000	u16RidLen	u16	read-only 0x12	Length of this RID including the length field
+0x0002	u8DriverName[16]	u8		The driver name and version can be written here for debugging support

Encapsulation Transformations

This section is only applicable to hosts that require ethernet style payloads. If the host has configured the card for LLC payloads, ie. unmodified payloads, (see OperatingMode in the General Configuration) then all payloads will pass through as is. 802.11 networks require 802.3 payload encapsulations. The PC4500/4800 will translate transmitted payloads and received payloads as required for ethernet hosts. (**NOTE**: when registering to an Aironet AP with Aironet extensions enabled, the PC4500/4800 will "adopt" the encapsulation rules in use by the AP -- this is done to prevent tables from being out of sync.)

Two methods of encapsulating ethertype packets on an 802.3 network are available:

- RFC 1042
- 802.1H

RFC1042 place the following LLC/SNAP in front of the payload: 0xAA 0xAA 0x03 0x00 0x00 0x00
802.1H places the following LLC/SNAP in front of the payload: 0xAA 0xAA 0x03 0x00 0x00 0xF8

When transmitting an ethernet packet onto the wireless medium:

1. if the ethertype field is a length (less than or equal to 0x5DC), then the ethertype field is stripped and the remaining payload (LLC) is used.
2. otherwise, translation is performed by scanning the list of ethertypes in the encapsulation RID. If a match is found, then the actions specify RFC1042 or 802.1H encapsulation. A zero ethertype ends the list and specifies the default actions when a match is not found.

When receiving a wireless packet:

1. if the packet is 802.1H, then the 802.1H header is stripped (back to ethertype)
2. if the packet is RFC1042, then the RFC1042 header is stripped only if the matching ethertype action or default action so specifies
3. if the packet is neither RFC1042 nor 802.1H, a length field is prepended to the packet

Offset	Name	Type	Factory default	Description
+0x0000	u16RidLen	u16	read-only 0x22	Length of this RID including the length field
+0x0002	EtherType	u16	0	Note, the ethertype values are in network transmission order. So IP (0x800) is actually (0x0008). Zero ends the list and selects the default action.
+0x0004	Action	u16	0	This field is bit encoded as follows: bit 0 (0x0001) 1=RFC1042 is kept for receive packets. bit 1 (0x0002) 0=RFC1042 is used for transmit encapsulation. 1=802.1H is used for transmit encapsulation.
…	Repeat above two fields		0	

The Encapsulation RID allows for 7 ethertypes to be entered plus a default action (ethertype = 0).

Capabilities RID

This RID indicates the capabilities of the radio.

Offset	Name	Type	Value	Description
+0x0000	u16RidLen	u16	read-only	Length of this RID including the length field
+0x0002	au8OUI	u8[3]	0x00 0x40 0x96	This field will give the manufacturer OUI (fourth byte always zero).
+0x0006	ProductNum	u8[3]	0x0004	This field will give the product number.
+0x0008	au8ManufacturerName	u8[32]	Aironet Wireless Communications	ASCIIz encoding of manufacturer name.
+0x0028	au8ProductName	u8[16]	PC4500 / 4800	ASCIIz encoding of product name.
+0x0038	au8ProductVersion	u8[8]	.	ASCIIz encoding of product (firmware?) version.
+0x0040	au8FactoryAddress	u8[6]		This field will contain the OEM assigned IEEE address. If there is no OEM address assigned, the Aironet assigned IEEE Address will be returned in this field.
+0x0046	au8AironetAddress	u8[6]		This field will contain the Aironet factory assigned IEEE address.
+0x004C	u16RadioType	u16	0x0001	This field will be bitmapped as follows. 0x01 = 802.11 FH 0x02 = 802.11 DS 0x04 = LM2000 (Legacy) DS Note, more than one bit may be set for radios supporting multiple modes of operation.
+0x004E	u16RegDomain	u16		This field indicates the registration domain/country. The values as assigned by 802.11 will be used.
+0x0050	au8Callid	u8[6]		This field indicates the callid assigned to the unit (if RegDomain is Japan) Each nibble will contain one decimal digit of the 12 digit callid. (Note, this is not the encoded format).
+0x0056	au8SupportedRates	u8[8]	0x02, 0x04, 0's	This field will indicate the 802.11 supported rates as specified in the rates.
+0x005E	u8RxDiversity	u8	0x03	This field will indicate the number of antennas supported as a bit mask.
+0x005F	u8TxDiversity	u8	0x03	This field will indicate the number of antennas supported as a bit mask.
+0x0060	au16TxPowerLevels	u16[8]	250	This table indicates the supported transmit power levels. (values are in mW) Zero terminates the list. Note, this may be further restricted depending on country selected.
+0x0070	u16HardwareVersion	u16	0	This indicates the revision of hardware.
+0x0072	u16HardwareCapabilities	u16	0	This is a bit-mapped field indicating harware capabilities. No bits have been assigned yet. Initially this is zero.
+0x0074	u16TemperatureRange	u16	0	This indicates the temperature range capability.
+0x0076	u16SoftwareVersion	u16	0	This indicates the revision of software. The upper byte indicates the major version and the lower byte the minor version.
+0x0078	u16SoftwareSubVersion	u16	0	This indicates the sub-revision of software.
+0x007A	u16InterfaceVersion	u16	0	This indicates the revision of the interface. This will be bumped whenever there are incompatible modifications made to the interface. This may be bumped on first release to ensure that "unreleased" utilities/drivers become unusable.
+0x007C	u16SoftwareCapabilities	u16[10]	0	This field gives a bit mapped indication of capabilities. No capability bits have yet been assigned.
+0x007E	u16BootBlockVersion	u16		This indicates the revision of bootblock software. The upper byte indicates the major version and the lower byte the minor version. Note, BCD encoding is used. (version 2.11 would be 0x0211.)

Status RID

This RID indicates the current status of the radio.

Offset	Name	Type	Description
+0x0000	u16RidLen	u16	Length of this RID including the length field
+0x0002	au8MacAddress	u8[8]	The MAC address in use by the station.
+0x0008	u16OperationalMode	u16	Bit-mapped. 0x0001 = Configured 0x0002 = MAC Enabled 0x0004 = Receive Enabled (to host) 0x0010 = In Sync (synchronized to cell) 0x0020 = Associated 0x8000 = Error
+0x000A	u16ErrorCode	u16	Non-zero if an error state has been entered
+0x000C	u16CurrentSignalQuality	u16	A measure of the current signal quality.
+0x000E	SSIDlength	u16	This length of the following SSID.
+0x0010	SSID	u8[32]	The SSID that is currently in effect. Note, this could be null if multiple SSIDs are valid and the unit is scanning.
+0x0030	au8ApName	u8[16]	The name of the current BSSID (ESS mode only)
+0x0040	au8CurrentBssid	u8[32]	BSSID that is currently in effect. Note, this could be broadcast if the unit is scanning.
+0x0046	au8PreviousBssid1	u8[6]	A former BSSID.
+0x004C	au8PreviousBssid2	u8[6]	A former BSSID.
+0x0052	au8PreviousBssid3	u8[6]	A former BSSID.
+0x0058	u16BeaconPeriod	u16 (kus)	The current beacon period.
+0x005A	u16DtimPeriod	u16	The current DTIM period (number of beacons between DTIMs).
+0x005C	u16AtimDuration	u16 (kus)	The current ATIM window duration. Adhoc/Ibss only
+0x005E	u16HopPeriod	u16 (kus)	The current hopping period.
+0x0060	u16ChannelSet	u16	The current channel set.
+0x0062	u16Channel	u16	The current operating channel.
+0x0064	u16HopsToBackbone	u16	0 indicates a backbone association.
+0x0066	u16ApTotalLoad	u16	Total load including broadcast/multicast from backbone. This is the value extracted from the Aironet element.
+0x0068	u16OurGeneratedLoad		Total load generated by our station (transmitted and received). Excludes received broadcast/multicast traffic.
+0x006A	u16AccumulatedArl	u16	

Radio Information RID

This RID contains radio specific information.

Access Point RID

This RID contains additional information required by the access point for operation.

Offset	Name	Type	Value	Description
+0x0000	u16RidLen	u16	0x06, read-only	Length of this RID including the length field
+0x0002	TIM Addr	u16	Read only	The “Traffic Indication Map” is updated by the host via the AUX I/O ports. This is the address of TIM.
+0x0004	Airo Addr	u16	Read only	The "Aironet Information Element" is updated by the host via the AUX I/O ports. This is the address of the Aironet Element.

Statistics RID

Statistics are available as both 16-bit and 32-bit structures. All the statistics structures are read-only, however, the delta statistics may be cleared using the 0xFF62 or 0xFF6A RID.

16-bit Offset	32-bit Offset	Field	Description
+0x0000	+0x0000	u16RidLen	Length of the RID including the length field.
---	+0x0002	u16Spacer	Spacer to align the u32 tallies.
+0x0002	+0x0004	Tal.RxOverrunErr	Receive overruns -- No buffer available to handle the receive. (result is that the packet is never received)
+0x0004	+0x0008	Tal.RxPlcpCrcErr	PLCP header checksum errors (CRC16).
+0x0006	+0x000C	Tal.RxPlcpFormat Err	PLCP format errors.
+0x0008	+0x0010	Tal.RxPlcpLength Err	PLCP length is incorrect.
+0x000A	+0x0014	Tal.RxMacCrcErr	Count of MAC CRC32 errors.
+0x000C	+0x0018	Tal.RxMacCrcOk	Count of MAC CRC32 received correctly.
+0x000E	+0x001C	Tal.RxWepErr	Count of all WEP ICV checks that failed. (this value is included in Tal.RxMacCrcOk)
+0x0010	+0x0020	Tal.RxWepOk	Count of all WEP ICV checks that passed. (this value is included in Tal.RxMacCrcOk)
+0x0012	+0x0024	Tal.RetryLong	Count of all long retries. (Does not include first attempt for a packet).
+0x0014	+0x0028	Tal.RetryShort	Count of all short retries. (Does not include first attempt for a packet).
+0x0016	+0x002C	Tal.MaxRetries	Count of number of packets that max-retried -- ie were never ACK'd.
+0x0018	+0x0030	Tal.NoAck	Count of number of times that ACK was not received.
+0x001A	+0x0034	Tal.NoCts	Count of number of timer that CTS was not received.
+0x001C	+0x0038	Tal.RxAck	Count of number of expected ACKs that were received.
+0x001E	+0x003C	Tal.RxCts	Count of number of expected CTS's that were received.
+0x0020	+0x0040	Tal.TxAck	Count of number of ACKs transmitted.
+0x0022	+0x0044	Tal.TxRts	Count of number of RTS's transmitted.
+0x0024	+0x0048	Tal.TxCts	Count of number of CTS's transmitted.
+0x0026	+0x004C	Tal.TxMc	LMAC count of multicast packets sent (uses 802.11 Address1).
+0x0028	+0x0050	Tal.TxBc	LMAC count of broadcast packets sent (uses 802.11 Addresss1). **
+0x002A	+0x0054	Tal.TxUcFrags	LMAC count of ALL unicast fragments and whole packets sent (uses 802.11 Address1).
+0x002C	+0x0058	Tal.TxUcPackets	LMAC count of unicast packets that were ACK'd (uses 802.11 Address 1).
+0x002E	+0x005C	Tal.TxBeacon	Count of beacon packets transmitted.
+0x0030	+0x0060	Tal.RxBeacon	Count of beacon packets received matching our BSSID.
+0x0032	+0x0064	Tal.TxSinColl	Transmit single collisions. **
+0x0034	+0x0068	Tal.TxMulColl	Transmit multiple collisions. **
+0x0036	+0x006C	Tal.DefersNo	Transmit frames sent with no deferral. **
+0x0038	+0x0070	Tal.DefersProt	Transmit frames deferred due to protocol.
+0x003A	+0x0074	Tal.DefersEngy	Transmit frames deferred due to energy detect.
+0x003C	+0x0078	Tal.DupFram	Duplicate receive frames and fragments.
+0x003E	+0x007C	Tal.RxFragDisc	Received partial frames. (each tally could indicate the discarding of one or more fragments)

+0x0040	+0x0080	Tal.TxAged	Transmit packets exceeding maximum transmit lifetime. **
+0x0042	+0x0084	Tal.RxAged	Receive packets exceeding maximum receive lifetime. **
+0x0044	+0x0088	Tal.LostSync.MaxRetry	Lost sync with our cell due to maximum retries occuring.
+0x0046	+0x008C	Tal.LostSync.MissedBeacons	Lost sync with our cell due to missing too many beacons.
+0x0048	+0x0090	Tal.LostSync.ArlExceeded	Lost sync with our cell due to Average Retry Level being exceeded.
+0x004A	+0x0094	Tal.LostSync.Deauthed	Lost sync with our cell due to being deauthenticated.
+0x004C	+0x0098	Tal.LostSync.Disassoced	Lost sync with our cell due to being disassociated.
+0x004E	+0x009C	Tal.LostSync.TsfTiming	Lost sync with our cell due to excessive change in TSF timing.
+0x0050	+0x00A0	Tal.HostTxMc	Count of multicast packets sent by the host.
+0x0052	+0x00A4	Tal.HostTxBc	Count of broadcast packets sent by the host.
+0x0054	+0x00A8	Tal.HostTxUc	Count of unicast packets sent by the host.
+0x0056	+0x00AC	Tal.HostTxFail	Count of host transmitted packets which failed.
+0x0058	+0x00B0	Tal.HostRxMc	Count of host received multicast packets.
+0x005A	+0x00B4	Tal.HostRxBc	Count of host received broadcast packets.
+0x005C	+0x00B8	Tal.HostRxUc	Count of host received unicast packets.
+0x005E	+0x00BC	Tal.HostRxDiscard	Count of host received packets discarded due to: Host not enabling receive. Host failing to dequeue receive packets quickly. Packets being discarded due to magic packet mode.
+0x0060	+0x00C0	Tal.HmacTxMc	Count of internally generated multicast (DA) packets.
+0x0062	+0x00C4	Tal.HmacTxBc	Count of internally generated broadcast (DA) packets.
+0x0064	+0x00C8	Tal.HmacTxUc	Count of internally generated unicast (DA) packets.
+0x0066	+0x00CC	Tal.HmacTxFail	Count of internally generated transmit packets that failed.
+0x0068	+0x00D0	Tal.HmacRxMc	Count of internally received multicast (DA) packets.
+0x006A	+0x00D4	Tal.HmacRxBc	Count of internally received broadcast (DA) packets.
+0x006C	+0x00D8	Tal.HmacRxUc	Count of internally received multicast (DA) packets.
+0x006E	+0x00DC	Tal.HmacRxDiscard	Count of internally received packets that were discarded (usually because the destination address is not for the host).
+0x0070	+0x00E0	Tal.HmacRxAccepted	Count of internally received packets that were accepted.
+0x0072	+0x00E4	Tal.SsidMismatch	Count of SSID mismatches.
+0x0074	+0x00E8	Tal.ApMismatch	Count of specified AP mismatches.
+0x0076	+0x00EC	Tal.RatesMismatch	Count of rate mismatches.
+0x0078	+0x00F0	Tal.AuthReject	Count of authentication rejections.
+0x007A	+0x00F4	Tal.AuthTimeout	Count of authentication timeouts.
+0x007C	+0x00F8	Tal.AssocReject	Count of association rejections.
+0x007E	+0x00FC	Tal.AssocTimeout	Count of association timeouts.
+0x0080	+0x0100	Tal.ReasonOutsideTable	Count of reason/status codes of greater than 19. (Values of 0 = successful are not counted)
+0x0082	+0x0104	Tal.ReasonStatus1	Unspecified reason.
+0x0084	+0x0108	Tal.ReasonStatus2	Previous authentication no longer valid.
+0x0086	+0x010C	Tal.ReasonStatus3	Deauthenticated because sending station is leaving (has left) IBSS or ESS.
+0x0088	+0x0110	Tal.ReasonStatus4	Disassociated due to inactivity

+0x008A	+0x0114	Tal.ReasonStatus 5	Disassociated because AP is unable to handle all currently associated stations.
+0x008C	+0x0118	Tal.ReasonStatus 6	Class 2 Frame received from non-Authenticated station.
+0x008E	+0x011C	Tal.ReasonStatus 7	Class 3 Frame received from non-Associated station.
+0x0090	+0x0120	Tal.ReasonStatus 8	Disassociated because sending station is leaving (has left) BSS.
+0x0092	+0x0124	Tal.ReasonStatus 9	Station requesting (Re)Association is not Authenticated with responding station.
+0x0094	+0x0128	Tal.ReasonStatus 10	Cannot support all requested capabilities in the Capability Information Field.
+0x0096	+0x012C	Tal.ReasonStatus 11	Reassociation denied due to inability to confirm that Association exists
+0x0098	+0x0130	Tal.ReasonStatus 12	Association denied due to reason outside the scope of the 802.11 standard.
+0x009A	+0x0134	Tal.ReasonStatus 13	Responding station does not support the specified Authentication Algorithm.
+0x009C	+0x0138	Tal.ReasonStatus 14	Received an out of sequence Authentication Frame.
+0x009E	+0x013C	Tal.ReasonStatus 15	Authentication rejected due to challenge failure.
+0x00A0	+0x0140	Tal.ReasonStatus 16	Authentication rejected due to timeout waiting for next frame in sequence.
+0x00A2	+0x0144	Tal.ReasonStatus 17	Association denied because AP is unable to handle additional associated stations.
+0x00A4	+0x0148	Tal.ReasonStatus 18	Association denied due to requesting station not supporting all basic rates.
+0x00A6	+0x014C	Tal.ReasonStatus 19	Reserved
+0x00A8	+0x0150	Tal.RxMan	Count of management packets received and handled.
+0x00AA	+0x0154	Tal.TxMan	Count of management packets transmitted.
+0x00AC	+0x0158	Tal.RxRefresh	Count of null data packets received.
+0x00AE	+0x015C	Tal.TxRefresh	Count of null data packets transmitted.
+0x00B0	+0x0160	Tal.RxPoll	Count of PS-Poll packets received.
+0x00B2	+0x0164	Tal.TxPoll	Count of PS-Poll packets transmitted.
+0x00B4	+0x0168	Tal.HostRetries	Count of long and short retries used to transmit host packets (does not include first attempt).
+0x00B6	+0x016C	Tal.LostSync.HostReq	Lost sync with our cell due to host request.
+0x00B8	+0x0170	Tal.HostTxBytes	Count of bytes transferred from the host.
+0x00BA	+0x0174	Tal.HostRxBytes	Count of bytes transferred to the host.
+0x00BC	+0x0178	Tal.ElapsedUsec	Total time since power up (or clear) in microseconds.
+0x00BE	+0x017C	Tal.ElapsedSec	Total time since power up (or clear) in seconds.
+0x00C0	+0x0180	Tal.LostSyncBetterAP	Lost Sync to switch to a better access point

Note, on an AP, not all of the above fields are updated since some of the packets are passed to the host and bypass the code that would update the counter.

Some counts can be derived:

TOTAL_RX_ERRORS =
Tal.RxOverrunErr + Tal.RxPlcpFormatErr + Tal.RxPlcpLengthErr + Tal.RxMacCrcErr

TOTAL_RX_OK =
Tal.RxMacCrcOk

TOTAL_RX_ATTEMPTS =
TOTAL_RX_ERRORS + TOTAL_RX_OK

MISC_RADIO_RX_DISCARDS =
Tal.SsidMismatch + Tal.ApMismatch + Tal.RatesMismatch + Tal.RxFragDisc

TOTAL_RETRIES =
Tal.RetryLong + Tal.RetryShort

PROTOCOL_OVERHEAD_RX_FRAMES =
Tal.RxMan + Tal.RxRefresh + Tal.Rx.Poll + Tal.RxBeacon + Tal.RxAck + Tal.RxCts + Tal.TxCts

**note a count of received RTS frames will be approximately equal to the number of CTS frames transmitted

PROTOCOL_OVERHEAD_TX_FRAMES=
Tal.TxMan + Tal.TxRefresh + Tal.TxBeacon

TOTAL_TX_HOLDOFFS =
Tal.DefersProt + Tal.DefersEngy

NUMBER_TX_FRAMES_REQUIRING_RETRIES =
Tal.TxSinColl + Tal.TxMulColl

The following calculated counts may be useful for debugging purposes:

TOTAL_RX_PLCP_OK =
Tal.RxPlcpFormatErr + Tal.RxPlcpLengthErr + Tal.RxMacCrcErr + Tal.RxMacCrcOk

TOTAL_RX_PLCP_ERROR_COUNT =
Tal.RxPlcpCrcErr

Some counts should approximate (??=??) each other:

Tal.TxAck ??=?? TOTAL_RX_OK

Tal.TxMan ??=?? Tal.TxPoll + Tal.TxRts + Tal.TxCts + Tal.TxAck

Power Save Operation

Several levels of power save are implemented in the PC4500/4800:

PSP	Should be used in the cases where small amounts of data are intermittently transferred (example = a handheld scanner used to read bar codes and use the information to query a database)
Fast-PSP	Can be used in the cases where large amounts of data are intermittently transferred. Will result in a more efficient use of battery as well as effecting a quicker data transfer. (example = pen based unit which updates information on a screen by screen basis)
	In both the above modes, the user may enter values for the following parameters which will affect the amount of time that the PC4500/4800 wakes to check for pending data: ListenInterval, FastListenInterval, ListenDecay, FastListenDelay, SleepForDTIMs.
	In addition, if the network supports any of the MAGIC Packet or WAKE-ON-LAN functions, the user may configure the PC4500/4800 to ignore all traffic until an appropriate wake-up frame is received.
	Also available is a Host Sleep command (can be used in conjunction with all modes) which is used to inform the PC4500/4800 that the host is not available. This feature allows the PC4500/4800 to power down the host interface which results in the lowest power save mode.

The PC4500/4800 product line offers the user many alternatives which can be used to conserve power when operating in a battery powered environment. The parameters are implemented in such a fashion as to define consistent power save operation and definition from the client perspective. This allows the client to roam between access points which may have different operating parameters while maintaining the same power consumption characteristics.

The parameters affecting power save operation are described in detail below.

ListenInterval = value in kusec used to set the maximum time that the client will wait before waking to listen for DTIM frames. This value is currently limited to be no more than 2 minutes. (This value is rounded to the nearest integer beacon interval as set on the current access point.)

FastListenInterval = value in kusec used to set the current listen interval to be used immediately after a receive operation. (This value is rounded to the nearest integer beacon interval as set on the current access point and can be no greater than the ListenInterval.)

ListenDecay = integer number of times to cycle using the current listen interval value before doubling the value. (The starting point is the FastListenInterval value)

FastListenDelay = value in kusec used to set the time to wait after a data transmit before resuming the power save operation. At the completion of this time, the FastListenInterval is started. This parameter is not used in Fast-PSP operation.

SleepForDTIMs = boolean parameter where 0 indicates to the psp client that it must wake for ALL DTIMs and for any beacons for which the ListenInterval will expire before the next expected DTIM. A nonzero value indicates that the psp client will wake only for beacons specified by the ListenInterval (this means that broadcast and multicast traffic may be missed).

In order to achieve maximal power saving in power save mode, some cooperation is required from the host. If the PC4500/4800 is aware that it will not be accessed by the host, it can achieve additional power saving. To achieve this, a Host Sleep command is issued by the host to the PC4500/4800. After issuing this command, the host must not access any PC4500/4800 registers until the host "awakens" the PC4500/4800. The PC4500/4800 will maintain the wireless network connection to receive any pending traffic. If any valid packets are received, then an interrupt will be generated to the host as normal. To service a receive

packet, or to transmit a packet, the host must first awaken the PC4500/4800 by setting EvAck.Awaken. This must be done twice without any delays:

```
void awaken_4500(void)
{
   push_interrupt_enable_state();
   disable_interrupts();
   OUT4500(EVACK, EV_AWAKEN);      // first awakens
   OUT4500(EVACK, EV_AWAKEN);      // second does actual write
   pop_interrupt_enable_state();
}
```

After some delay, which could be 50 milliseconds or more, the PC4500/4800 will issue an EvStat.Awake event. The host should then acknowledge this (EvAck.Awake) and proceed to processing the actual event, or transmitting the required packet.

NOTE: THE FOLLOWING PSUEDO CODE ONLY APPLIES TO INFRASTRUCTURE MODE.

```
Psp.CurListenInterval = FastListenInterval
Psp.CurDecayValue = ListenDecay

while (1) {
  wait until allowed to sleep;
  // allowed to go back to sleep as soon as there is no traffic pending to us (no directed or multicast)
  // and we have nothing to transmit -- there is no minimum time that we must stay awake for
  while (not allowed to sleep) {
    if (receive directed packet) Psp.CurListenInterval = 0;
    if (transmit packet) Psp.CurListenInterval = 0;
  }

  // note, receiving or transmitting will set Psp.CurListenInterval to zero
  if (Psp.CurListenInterval == 0) {
    // use FastListenInterval since we transmitted or received...
    Psp.CurListenInterval = FastListenInterval;
    Psp.CurDecayValue = ListenDecay;
  } else {
    if (--Psp.CurDecayValue == 0) {
      // double the current listen interval and limit to the user input ListenInterval
      Psp.CurListenInterval += Psp.CurListenInterval;
      if (Psp.CurListenInterval > ListenInterval)  Psp.CurListenInterval = ListenInterval;
      Psp.CurDecayValue = ListenDecay;
    }
  }

  timeToWakeUp = currentTime + Psp.CurListenInterval;

  timeOfWakeUpBeacon = calculate time of beacon following timeToWakeUp;

  if (SleepForDtim == 0) {
    // may have to wake up sooner for a DTIM to receive broadcast traffic
    timeOfNextDtim = calculate time of next dtim;

    // take the sooner of the next dtim or the wakeup beacon
    if (timeOfNextDtim < timeOfWakeupBeacon) timeOfWakeUpBeacon = timeOfNextDtim;
  }

  Sleep until timeOfWakeUpBeacon              // do the sleep for the listen interval or until dtim

          resynchronize to cell....
}
```

EXAMPLE

```
FastListenInterval = 100,
ListenInterval = 2000,
ListenDecay = 2,
```

then assuming no traffic the listen times would be
-- 100, 100, 200, 200, 400, 400, 800, 800, 1600, 1600, 2000, 2000, 2000, 2000, ...

8

CHAPTER 8

PLAP Serial Client Software Interface

This chapter details the Peripheral Link Access Protocol (PLAP) software interface to the PC4500/4800 when using serial mode.

Introduction

PLAP is a half-duplex formalized handshaking method of transferring data. The PC4500/4800 processor connects like a peripheral device to a host PC's asynchronous serial port. The serial connection must operate at a single standard baud rate in the range of 7200bit/s to 115.2kbit/s. The interface is designed to be symmetrical, that is, it will look the same from either end of the connection. Thus, the PC4500/4800 and the driver on a host PC will use the same frame formats to communicate with each other.

Serial port settings: 8 bits, No-parity, 2-stop bits
Supported rates: 7.2, 9.6, 14.4, 19.2, 28.8, 38.4, 57.6, 76.8, 115.2 kbit/s

Note: The PC4800 is limited to 2 Mb/s or less as a radio supported rate in serial mode.

Media Access Control

Communication between PLAP layers on a PC4500/4800 and the host PC takes on the form of data and control frames transmitted over a peripheral line. Media access is simply "transmit if ready and only if media is not busy." "Busy" is defined as the BUSY_IN line of the serial interface being active (host PC perspective). There is no immediate confirmation of frame delivery. Once the closing byte of the frame has been sent, transmission is assumed successful. The nature of a cabled serial interface is such that it

can be made reliable by design since the cable lengths are short. The normal transport layer protocols used in network operating systems are therefore sufficient for reliable transport of application frames. A host application (or driver) may choose to occasionally send a SYNC_REQ frame to verify cable connectivity still exists by the receipt of a SYNC_ACK frame.

Note:

BUSY_OUT line controlled by host PC
BUSY_IN line controlled by PC4500/4800

Autobaud Sequence

In order for the PC4500/4800 to determine which baud rate that the host PC is using, a known sequence of characters must be transmitted from the host PC to the PC4500/4800. Upon correct recognition of this sequence, the PC4500/4800 will fix its baud rate to the identified rate. The autobaud sequence is <CR><CR><CR><3> (<0x0d><0x0d><0x0d><0x33>).

PLAP Synchronization

Two PLAP layers connected by a peripheral line must ensure that they have frame synchronization in both directions. Since there is no special character used to delineate frame boundaries, correct synchronization is determined by the successful reception of frames and by the handshake lines of the interface.

Synchronization should be verified immediately after the autobaud sequence has been sent. Synchronization should also be checked periodically (every 5 - 60 seconds) if no data has been received from the remote end during that interval. This check serves as a double check that the other end is still there and active.

To verify synchronization, a host node sends a SYNC_REQ frame and waits for the SYNC_ACK frame to be received. If the SYNC_ACK frame is not being received within 1 second, another SYNC_REQ should be sent. Normally, the PC4500/4800 sends a SYNC_ACK immediately, however, there may be other processes running which can delay the response. (This is true of host devices in which operating systems occasionally take time to store its current state and as well as the PC4500/4800 may be performing a higher priority task and not able to immediately begin a serial transmission.)

Receiving/Transmitting Frames

To transmit a frame the host must first check the BUSY_IN line. If the BUSY_IN line is not active, the host must first activate the BUSY_OUT line before transmitting the PLAP frame. The host must keep the BUSY_OUT line active while transmitting the PLAP frame and deactivate the BUSY_OUT line after the frame is transmitted. On reception of frames, the host should accept the frames only if BUSY_IN is active. If the BUSY_IN line becomes inactive at any time other than the end of a frame, the frame reception should be aborted and PLAP re-armed to wait for the start of the next frame.

PLAP Frame Format

The PLAP frame starts with a Frame Type character followed by the frame length. After the frame length can be several fields/parameters of data. Immediately following the data is the checksum and the closing character.

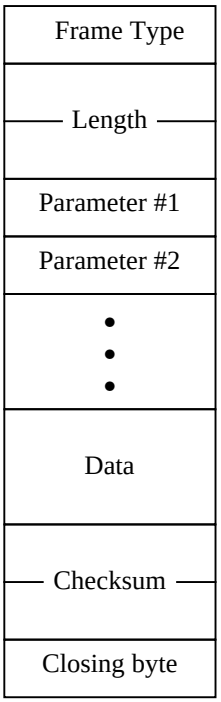


Figure 8.1 - PLAP Packet Format

The PLAP frame consists of the following fields.

- The first field is Frame Type which identifies the type of frame. Several types are defined and described in the next sections. The Most Significant Bit (MSB) is used to indicate if the checksum field is operational (see checksum field description below).
  MSB = 0; checksum field is valid
  MSB = 1; checksum field is not valid
- Following Frame Type is the two-byte Length field. The Length field contains the total length of the frame from the Frame Type character to the Closing character inclusive. The length is sent high byte first.
- Following the Length field are the set of parameters, if any, that each frame type requires. The number of parameters can be different for each type. Any data associated with the frame type immediately follows the last parameter. This can be of any length from 0 to 1514 characters.
- Following the data is a two-byte checksum which is the inclusive addition of all characters in the frame excluding the checksum and closing byte. Note: MSB of frame type must be 0 if this field is valid.
- The closing character is simply a defined character used as another verification that framing was maintained in the packet. The character used is the ASCII EOT character 0x04.

Note: all word or long word fields are sent big endian (Motorola format) MSB to LSB with the high byte sent first. Character data is treated as a byte stream.

Migrating from the LM2000 to the PC4500/4800

The PC4500/4800 supports the same PLAP frame format as the LM2000. The SYNC_REQ, SYNC_ACK, DATA and DOWNLOAD frames are identical between the two products. In order to take full advantage of the PC4500/4800, the user must support the new HOST_COMMAND and COMMAND_RESPONSE frame types.

A minimal requirement for migration is the implementation of the use of two stop bits. The PC4500/4800 serial interface does not default to any given baud rate and therefore must 'learn' which rate is being used by the host.

PLAP Commands

Table 8-1 lists the valid PLAP commands.

Table 8-1. PLAP Commands

FRAME TYPE (hex)	NAME	DIRECTION	DESCRIPTION
00 / 80	SYNC_REQ	Host to PC4500/4800	Verify connectivity
01 / 81	SYNC_ACK	PC4500/4800 to Host	Response to SYNC_REQ frame
02 / 82	DATA	Both	Data frames
04 / 84	CONFIGURE	Host to PC4500/4800	Used to perform quick configuration
07 / 87	DOWNLOAD	Host to PC4500/4800	Used to upgrade firmware
10 / 90	HOST_COMMAND	Host to PC4500/4800	Issue host commands
11 / 91	COMMAND_RESPONSE	PC4500/4800 to Host	Response to host command

SYNC_REQ

Frame Type: 0x00, 0x80

```
struct sync_req_frm {
    unsigned char    Frame_type;    // 0x00, 0x80
    unsigned short   Length;        // 0x0006
    unsigned short   Checksum;      // 0x0006, 0x0086
    unsigned char    EOT;           // 0x04
    };
```

The sync frames are used to verify connectivity of the peripheral interface. Each PLAP implementation will assume that its receiver is synchronized and attempt to verify that its transmitter is synchronized with the remote receiver. The PLAP process on the PC4500/4800 will immediately transmit a SYNC-ACK frame in response to a SYNC-REQ frame.

Compatibility: PC4500/4800 and LM2000

SYNC_ACK

Frame Type: 0x01, 0x81

```
struct sync_ack_frm {
    unsigned char     Frame_type;        // 0x01 , 0x81
    unsigned short    Length;            // 0x0006
    unsigned short    Checksum;          // 0x0007, 0x0087
    unsigned char     EOT;               // 0x04
    };
```

The SYNC_ACK frame is sent in response to a SYNC_REQ frame. This frame is always sent before any other frame that may be pending.

Compatibility: PC4500/4800 and LM2000

DATA

Frame Type: 0x02, 0x82

```
struct data_frm {
    unsigned char     Frame_type;        // 0x02, 0x82
    unsigned short    Length;            // xxxx
    unsigned char     Destination[6];
    unsigned char     Source[6];
    unsigned short    Type_lgth;
    unsigned char     Data[xxxx];
    unsigned short    Checksum;          // xxxx
    unsigned char     EOT;               // 0x04
    };
```

Data frames are sent using this frame type. An Ethernet or 802.3 frame, excluding the preamble and CRC fields, is encapsulated within a PLAP frame. When the frame gets routed to an Ethernet backbone LAN, it will appear exactly as contained within this frame.

The Destination field is a six-byte IEEE-802 style address and is the ultimate destination of the frame. The Source field contains the six-byte ultimate source address of the frame, which would contain the address of the PC4500/4800 when the DATA frame originates from the host device. The host device should use the same address as its PC4500/4800. The MAC address can be found by issuing a HOST_COMMAND frame.

The Type/Length field contains an Ethernet type or 802.3 Length field for the frame.

The Data field contains any network protocol headers and data that are required by the application. The data field contains the data being transferred. The maximum length for this field is 1500 bytes.

Compatibility: PC4500/4800 and LM2000

CONFIGURE

Frame Type: 0x04, 0x84

Configure type 0x10:

```
                                                    // Default Values
struct config16_frm {
    unsigned char     Frame_type;           // 0x04, 0x84
    unsigned short    Length;               // 0x005B
    unsigned char     Type;                 // 0x10
    unsigned short    SSIDlen;              // 0x0007
    unsigned char     SSID[32];             // 'tsunami'
    unsigned char     SupportedRates[8];    //
    unsigned char     StationMacId[6];      // 00:00:00:00:00:00
    unsigned char     NodeName[16];         // 0's
    unsigned short    PowerSaveMode;        // 0 = CAM
    unsigned short    ScanMode;             // 0
    unsigned short    DsChannel             // 0 = lowest channel
    unsigned char     Reserved[14];         // reserved for future use
    unsigned short    Checksum;             // xxxx
    unsigned char     EOT;                  // 0x04
    };
```

This Configure frame is only sent from the host device to the PC4500/4800. It is meant to be a 'quick configuration' of the PC4500/4800. The parameters are the minimum needed to establish a connection in infrastructure mode. The firmware defaults will be used for all other parameters.

Compatibility: PC4500/4800

DOWNLOAD

Frame Type: 0x07 / 0x87

The PLAP download frame type has several subtypes used to perform a firmware upgrade.

Subtypes:

0x01	MODE_REQ
0x81	MODE_RESP
0x02	SET_MODE
0x03	PARAM_REQ
0x83	PARAM_RESP
0x04	ERASE_REQ
0x84	ERASE_RESP
0x05	PROG_REQ
0x85	PROG_RESP
0x07	ENABLE_REQ
0x87	ENABLE_RESP

Download type 0x01:

```
                                        // Default Values
struct mode_request_frm {
    unsigned char     Frame_type;       // 0x07, 0x87
    unsigned short    Length;           // 0x0007
    unsigned char     Type;             // 0x01
    unsigned short    Checksum;         // 0x000F, 0x008F
    unsigned char     EOT;              // 0x04
    };
```

Download type 0x81:

```
                                        // Default Values
struct mode_response_frm {
    unsigned char     Frame_type;       // 0x07, 0x87
    unsigned short    Length;           // 0x0009
    unsigned char     Type;             // 0x81
    unsigned char     Current_mode;     // 0x01=normal;
                                        0x02=upgrade mode
    unsigned char     Firmware_okay;    // 0x00=okay
    unsigned short    Checksum;         // xxxx
    unsigned char     EOT;              // 0x04
    };
```

Download type 0x02:

```
                                        // Default Values
struct set_mode_frm {
    unsigned char     Frame_type;       // 0x07, 0x87
    unsigned short    Length;           // 0x0008
    unsigned char     Type;             // 0x02
    unsigned char     New_mode;         // 0x02=upgrade mode
    unsigned short    Checksum;         // 0x0013, 0x0093
    unsigned char     EOT;              // 0x04
    };
```

Download type 0x03:

```
                                        // Default Values
struct params_request_frm {
    unsigned char     Frame_type;       // 0x07, 0x87
    unsigned short    Length;           // 0x0007
    unsigned char     Type;             // 0x03
    unsigned short    Checksum;         // 0x0011, 0x0091
    unsigned char     EOT;              // 0x04
    };
```

Download type 0x83:

```
                                                    // Default Values
struct params_response_frm {
    unsigned char     Frame_type;                   // 0x07, 0x87
    unsigned short    Length;                       // 0x000F
    unsigned char     Type;                         // 0x83
    unsigned char     Boot_vmjr;
    unsigned char     Boot_vmnr;
    unsigned char     Oper_vmjr;
    unsigned char     Oper_vmnr;
    unsigned char     Man_code;
    unsigned char     Dev_code;
    unsigned char     Dload_ksize;                  // 0x01
    unsigned char     Opfrm_kaddr;                  // 0x00
    unsigned short    Checksum;                     // xxxx
    unsigned char     EOT;                          // 0x04
    };
```

Download type 0x04:

```
                                                    // Default Values
struct erase_resquest_frm {
    unsigned char     Frame_type;                   // 0x07, 0x87
    unsigned short    Length;                       // 0x000C
    unsigned char     Type;                         // 0x04
    unsigned char     Reserved;                     // 0x00
    unsigned short    Erase_kaddr;                  // 0x0000
    unsigned short    Erase_ksize;                  // 0x0000
    unsigned short    Checksum;                     // 0x0017, 0x0097
    unsigned char     EOT;                          // 0x04
    };
```

Download type 0x84:

```
                                                    // Default Values
struct erase_response_frm {
    unsigned char     Frame_type;                   // 0x07, 0x87
    unsigned short    Length;                       // 0x0008
    unsigned char     Type;                         // 0x84
    unsigned char     Status;                       // 0x00=successful
    unsigned short    Checksum;                     // xxxx
    unsigned char     EOT;                          // 0x04
    };
```

Download type 0x05:

```
                                        // Default Values
struct prog_request_frm {
    unsigned char     Frame_type;       // 0x07, 0x87
    unsigned short    Length;           // 0x040C
    unsigned char     Type;             // 0x05
    unsigned char     Prog_ksize;       // 0x01
    unsigned short    Prog_kaddr;       // packet no. (0, 1, 2, …)
    unsigned short    Prog_chksum;      // 0x0000
    unsigned char     Data[1024];
    unsigned short    Checksum;         // xxxx
    unsigned char     EOT;              // 0x04
    };
```

Download type 0x85:

```
                                        // Default Values
struct prog_response_frm {
    unsigned char     Frame_type;       // 0x07, 0x87
    unsigned short    Length;           // 0x0008
    unsigned char     Type;             // 0x85
    unsigned char     Status;           // 0x00=successful
    unsigned short    Checksum;         // xxxx
    unsigned char     EOT;              // 0x04
    };
```

Download type 0x07:

```
                                        // Default Values
struct enable_request_frm {
    unsigned char     Frame_type;       // 0x07, 0x87
    unsigned short    Length;           // 0x0007
    unsigned char     Type;             // 0x07
    unsigned short    Checksum;         // 0x0015, 0x0095
    unsigned char     EOT;              // 0x04
    };
```

Download type 0x87:

```
                                        // Default Values
struct enable_response_frm {
    unsigned char     Frame_type;       // 0x07, 0x87
    unsigned short    Length;           // 0x0008
    unsigned char     Type;             // 0x87
    unsigned char     Status;           // 0x00=successful
    unsigned short    Checksum;         // xxxx
    unsigned char     EOT;              // 0x04
    };
```

A binary file format is used to download/program the PC4500/4800. The data is sent in 1024 (1K) byte blocks, starting from the beginning of the file. No swapping of the data portion is necessary.

The proper sequence used to perform a firmware upgrade is to:

1. Send the mode request.
2. Receive the mode response.
3. Send the parameter request.
4. Receive the parameter response.
5. Issue the set mode command (mode = 2 for firmware download).
6. Issue the erase flash command.
7. Receive the erase confirm response.
8. Issue sequential program block frames until complete binary file is transferred.
9. Alternately receive program block responses.
10. Issue the firmware enable command.
11. Receive firmware enable response.
12. Issue autobaud sequence and follow normal boot sequence.

Note: steps 1- 4 are optional and can be used to determine the current firmware version before continuing with the complete download.

Compatibility: PC4500/4800 and LM2000

HOST_COMMAND

Frame Type: 0x10, 0x90

```
                                        // Default Values
 struct host_command_frm {
     unsigned char     Frame_type;      // 0x10, 0x90
     unsigned short    Length;          // xxxx
     unsigned char     Type;            // 0x??
     unsigned short    Command;
     unsigned short    Parameter0;
     unsigned short    Parameter1;
     unsigned short    Parameter2;


     struct {                           // Optional
                       RID              // required for the Write
     };                                 // RID and Transmitter
                                        // Testing commands only

     unsigned short    Checksum;        // xxxx
     unsigned char     EOT;             // 0x04
     };
```

The HOST_COMMAND frame is sent to issue host commands to the PC4500/4800.

The “Type” field determines the type of command being issued.
Valid values are:

0x00	Issue a command
0x01	Setup PC4500/4800 error reporting

When using Type=0x00, the “Command” parameter must be one of the following values.

0x0010	NOP
0x0001	Enable
0x0002	Disable
0x0003	Lose Sync
0x0004	Soft Reset
0x0008	Read Configuration
0x0021	Read RID
0x0108	Write Configuration
0x0121	Write RID (RID structure is used to pass the info) Refer to chapter 7 for RID structures.
0x003E	Set PHY registers
0x003F	Transmitter testing (RID structure is used to pass the info) Refer to Transmitter testing below.

Refer to chapter 7 for more information on issuing host commands section on the parameter fields and values.

The PLAP transmitter testing command must be issued in the following format:

```
typedef struct host_command_frm {
    Frame_type                  = 0x10, 0x90;
    Length                      = xxxx;
    Type                        = 0x00;
    Command                     = 0x003F;
    Parameter0                  = cmd_block_lgth ;
    Parameter1                  = freq_block_lgth;
    Parameter2                  = pattern_block_lgth;
    CmdBlock[cmd_block_lgth]    = list of commands;
    FreqBlock[freq_block_lgth]  = list of frequencies;
    PatternBlock[pattern_block_lgth] = list of patterns;
    Checksum                    = xxxx;
    EOT                         = 0x04;
    } TXTESTFRM;
```

When using Type=0x01, the “Command” parameter is interpreted as a bit map field according to the following table.

Bit0	1=report PLAP frame errors, 0=disabled
Bit1	1=report command responses, 0=disabled
Bit2-6	Not used
Bit7	Reserved

Compatibility: PC4500/4800

COMMAND_RESPONSE

Frame Type: 0x11, 0x91

```
                                                    // Default Values
 struct command_response_frm {
      unsigned char     Frame_type;                 // 0x11, 0x91
      unsigned short    Length;                     // xxxx
      unsigned char     Type;                       // 0x??
      unsigned short    Status;
      unsigned short    Response0;
      unsigned short    Response1;
      unsigned short    Response2;

      struct {                                      // Optional
                        RID                         // returned on a Read RID
      };                                            // command only

      Unsigned short    Checksum;                   // xxxx
      Unsigned char     EOT;                        // 0x04
      };
```

The COMMAND_RESPONSE frame is sent in response to a HOST_COMMAND frame. This frame will also be sent in response to a PLAP frame error when error reporting is enabled.

Type: This field indicates type of response.

The "Type" field determines the type of response being returned.
Valid values are:

0x00	Response to an issued command The "Status" field and "Response" fields will contain values as indicated in chapter 7.
0x01	Response to a PLAP frame error. Status field: 0x7F00 Response0 field: 0x0000 unknown frame type received 0x0001 frame length too large 0x0002 Checksum invalid 0x0003 EOT character not received 0x0004 Allocate failed 0x0005 Unknown subtype 0x0006 Frame length too short 0x0007 Error receiving frame 0x0008 Receive buffer overrun Reponse1 field: 0x0000 Response2 field: 0x0000

Compatibility: PC4500/4800

PLAP Boot Strap of PC4500/4800

The following is the correct start sequence for PLAP mode.

1. Wait for Card Detect signals to go active (active LOW). ***
2. Enable power (VCC) to the socket. Set VPP1 = 5V, VPP2 = 0V.
3. Delay for power to stabilize (200 ms).
4. Assert PCMCIA reset.
5. Release PCMCIA reset.
6. Delay at least 10us.
7. Wait for Card Ready to go active. Alternately, delay 3 seconds.
8. Send autobaud sequence at selected baud rate (0x0D,0x0D,0x0D,0x33). When the autobaud sequence has been detected, BUSY_IN will be asserted.
9. If BUSY_IN is not detected within 1 second, repeat from step 8, possibly at a different baud rate.
10. If BUSY_IN is not detected within 3 repetitions of steps 8-9, repeat from step 4.
11. If BUSY_IN is not detected within 3 repetitions of steps 4-10, report ERROR.
12. Wait for 2 seconds for BUSY_IN to be deasserted. If it is not deasserted, repeat from step 4.
13. If BUSY_IN is not deasserted within 3 repetitions of steps 4-12, report ERROR.
14. Proceed with transmit of PLAP SYNC_REQUEST frame while obeying the BUSY_IN/BUSY_OUT protocol.
15. Wait for 2 seconds for receipt of SYNC_ACK frame.
16. If no SYNC_ACK is received, repeat from step 14.
17. If no SYNC_ACK after three SYNC_REQUEST attempts, report ERROR.

*** Assumes that cards can be removed. The PC4500/4800 connects the card detect signals to ground, so that if the host hardware would have a pull-up on this signal and a capability to test it, card detection would be possible.

PART 5

9

CHAPTER 9

OEM Radio Approval Information

This chapter contains information about the approvals, regulations, labeling requirements and configuration of the PC4500/4800 radio modules for operation in various countries.

Approvals

Safety

The PC4500/4800 is designed to meet the requirements of UL, CSA, VCCI and is compliant to the European Low Voltage Directives. However, the OEM is responsible for the individual safety agency approval of the entire OEM product. If necessary, Aironet will furnish any required information directly to the agency for the approval.

The PC4500/4800 is compliant to ANSI C95.1 1991 and the FCC OET-65 Standards for Safety Levels with Respect to Human Exposure to Radiated Frequency Electromagnetic Fields, 3kHz to 300GHz.

FCC Approvals (US and Territories)

The PC4500/4800 has full modular approval from the FCC under 15.247 of the FCC rules with a 2.2dBi dipole antenna 0 dBi snap on antenna, 12 dBi omni, 13.5 dBi Yagi, 19 dBi patch, and the 23 dBi parabolic dish. This certification applies to operation in the United States and its territories (Guam, American Samoa, Puerto Rico, and US Virgin Islands). The FCC ID for the PC4500 is LOZ102034, and for the PC4800, the FCC ID is LOZ102035.

The OEM is responsible for the overall compliance of the OEM product (unless it is installed in a PCMCIA slot in a laptop type device with the snap on antenna) to the FCC rules as stated in CFR 47 Part 15 (1998). The OEM is responsible for manufacturing, installing, and operating this equipment in

compliance to CFR 47 Parts 2 and 15. Aironet's FCC approval covers the radio and approved antennas only. The use of antennas not approved by the FCC for use with the Aironet radio is in violation of the FCC rules.

Using FCC approved antennas
If the OEM is using an Aironet FCC approved antenna, the OEM will be required to test the entire OEM product to Part 15 Subpart B for unintentional radiators. No additional FCC application or paperwork will be required concerning the radio itself. The OEM is responsible for the product meeting all applicable FCC standards.

Using third-party antennas
If the OEM is using a third-party antenna, Aironet Engineering will review the antenna specifications to determine if it will be covered under the family group of approved antennas. If the antenna falls under the family approval, then Aironet Engineering will issue a letter stating that the antenna is covered under the family approval and Aironet will add the information to our files.

If the antenna does not fall under the family approval, a FCC Class II Permissive Change or a FCC re-certification is required before it can be used with our radio. The Class II Permissive Change must be done through Aironet Engineering or our authorized agent. Contact your sales agent for further information. The FCC re-certification process is the responsibility of the OEM with the support of Aironet Engineering. Aironet will provide any necessary non-confidential information to the test lab and any confidential information directly to the FCC.

DOC Approvals (Canada)

The PC4500/4800 is certified for use in accordance with RSS-210 and RSS-139-1. Spread spectrum systems in the 2.4GHz range operating completely indoors or outdoors above 2450MHz do not require licensing. Those systems operating partially or completely outdoors below 2450MHz may require a license to operate. Additional information can be found in the CPC-2-0-01 Guidelines for Submission of Applications and the IC 2365BB Application for License to Install and Operate a Radio Transmitter. The Canadian Radio Approval number for the PC4500 is 2311 102 896A. For the PC4800, the approval number is 2311 391 115A.

If the OEM wishes to use an Aironet approved radio and antenna combination, the OEM will be required to have the entire OEM product tested, but no formal application for approval is necessary except "multi-listing" of the radio for use in the OEM product. If product qualification (EMC approval) is required, the OEM will submit his application to the DOC and reference the appropriate Aironet file. Aironet will interface with the Canadian authorities to update the files with the proper information for certification.

The OEM has the option of submitting his own application to the DOC in which case Aironet will provide information on the radio and antenna to the DOC. Any future changes to the OEM product can be done without contacting Aironet since the product registration will be in the OEM name.

If the OEM is using a third-party antenna, the OEM will be required to file the test report and application with the DOC on the OEM product, and Aironet will send any necessary confidential information directly to the Canadian DOC.

ETSI Approvals (Europe)

General Information
Based on the recent changes in the interpretation of the certification of radio modules by the European Union, the PC4500/4800 will now have modular approval. The maximum power rating for ETSI is 100mW ERIP.

Depending on the product and antenna used, the installation of the radio may or may not require the OEM to obtain a separate radio type approval certificate for the OEM product. It is the OEM's

responsibility to contact the European local authorities, Competent Body or Notified Body test lab to determine what applicable standards need to be applied for the product. The OEM can obtain the appropriate approval numbers and copy of certifications of the radio from their Aironet Sales agent.

The OEM will be required to test the OEM product with the radio installed to the requirements of EN 55022, EN 50082-1 and the Low Voltage Directive. The PC4500/4800 is designed to meet EN 55022 Class B levels. However, any additional shielding or filtering required to bring the overall product into compliance is the responsibility of the OEM.

If the customer must obtain a separate type approval number, the product will be required to be tested in a European lab for both ETS 300.328 and ETS 300.826. The Low Voltage Directive testing does not need to be done in Europe. Aironet will send the required confidential information directly to the authorities and reference the OEM's type approval test file number.

300.328 Type Approval
This is the actual test of the radio and antenna system for approval to be used in the European Radio Spectrum. The lab will do one complete test and then prepare the necessary number of reports that are required for submission in each country to obtain approval. The test labs can provide the address and contacts for each country's regulatory authority. Type approval can take from 5 weeks to 8 months or more depending on the country.

ETS 300.826 CE Mark Approval
Depending on the product, the Notified Body lab will test the entire OEM product to the required EMC standard. The lab will then submit one report to the local authority, which will then issue the CE mark for the product that will be accepted in the other EC countries. The tests can include the following but are not limited to:
EN 55022 Line Conducted
EN 1000-4-2 Electrostatic Discharge
EN 1000-4-3 RF Field Immunity (80MHz to 1GHz)
EN 1000-4-4 Electrical Fast Transient
EN 1000-4-5 Surge / Transient
EN 1000-4-6 Line Conducted Immunity (150kHz to 80MHz)
EN 1000-4-11 Voltage Transients
EN 6001-3-2 Power Line Harmonics
EN 6001-3-3 Voltage Flicker

EN 60950 Low Voltage Directive
This series of tests can be done in the US or Canada. It is recommended but not necessary to have it done at a UL or CSA approved lab. Upon completion of the testing, file a copy with the local European representative.

For More Information
To obtain the ETSI standards or for more information contact the following organizations at the phone numbers given:

Competent Authorities (Approval Agency)

Radiocommunications Agency (UK)
44 1712 110211

Notified Body Test Labs

RFI (UK)
44 1256 851193

Telefication (The Netherlands)
31 85 7807 80

OEM Labeling Requirements

US and US Territories

In accordance with FCC rules, the OEM product must have the radio approval number on the product label. This can be a second label added to the outside of the product. The label wording for the PC4500 should be as follows:

This product contains Aironet Radio Module
FCC ID: LOZ102034

The labeling for the PC4800 should be:

This product contains Aironet Radio Module
FCC ID: LOZ102035

Canada

In accordance with Canadian DOC rules, the OEM product must have the radio approval number on the product label. This can be a second label added to the outside of the product. The label wording for the PC4500 should be as follows:

This product contains Aironet Radio Module
Canada 2311 102 896A

The labeling for the PC4800 should be:

This product contains Aironet Radio Module
Canada 2311 391 115A

Europe

In compliance with the European Telecommunication Standards and European Authorities, the OEM product must have the radio approval number on the product label. This can be a second label added to the outside of the product. The label wording should be as follows:

This product contains Aironet Radio Module
Place the appropriate country identification number and country symbols here

Other Countries

The product must be labeled in accordance with the requirements of the specific country.

AWCSETEE Operation

A DOS-based utility called AWCSETEE is available for setting the EEPROM in the PC4500/4800 to specify the country of operation. Different countries have different sets of frequencies that are allowed in the 2.4GHz band. In order to operate on the correct frequencies, the country code must be set in the EEPROM that will tell the firmware which frequencies to operate on. When the PC4500/4800 is ordered, it is programmed to operate in the country to which it will be shipping. If it becomes necessary to change the country of operation from the original setting, the AWCSETEE utility must be run.

The following is a partial list of what is displayed when AWCSETEE is run with no command line options:

```
USAGE:     AWCSETEE [-d] [country]

     -d          - display information about the PC4500/4800
     -power      - limit transmit power, value in milliwatts
                         (0 removes limit)
     country     - country of operation from one of the following
                 NA (North America)
                 ETSI (Europe except Spain/France)
                 JAPAN
                 SPAIN
                 FRANCE
                 ISRAEL
     -h          Displays help information about AWCSETEE
```

The AWCSETEE utility can be used to just display the radio's current EEPROM contents by running it with the -d parameter. The command line would be:

```
awcsetee -d
```

To set the PC4500/4800 to run on the North American channel set, use the following command line:

```
awcsetee NA
```

To set the PC4500/4800 to run on the ETSI channel set, use the following command line:

```
awcsetee ETSI
```

To set the transmit power on the PC4500/4800, enter the value in milliwatts. For a 100 milliwatt transmit output, one would enter:

```
awcsetee –power 100
```

Appendix A – PC4500/4800 Troubleshooting

The PC4500/4800 provides LED messages and error codes. This chapter provides the general procedures for correcting common problems encountered when installing the PC4500/4800 system.

Indicator LEDs

The PC4500/4800 has two indicator LEDs located on the face of the card: one green and one amber.

The Green LED is the Link Integrity/Power LED. It glows when the card is receiving power and flashes when the PC4500/4800 is linked with the network.

The Amber LED is the Link Activity LED. It flashes when the PC4500/4800 is receiving or transmitting data, or flashes in a pattern to indicate an error condition.

See Tables A.1, A.2, and A.3 for an explanation of the LED Messages and Error Codes.

Table A.1 - Green LED Operating Messages

Green LED	Condition
Off	No power or error
Flashing Quickly	Power on, self-test OK, scanning for a network
Flashing Slowly	Associated with an infrastructure network

Table A.2 - Amber LED Operating Messages

Amber LED	Green LED	Condition
Flashing	Flashing Slowly	PC4500/4800 is transmitting or receiving data while associated with an access point
Flashing in a Pattern	Off	Indicates an error condition
Dim	Off	Card is not operating (reset)

Table A.3 - LED Error Codes

Amber LED	Condition
Blink at 2 second rate	RAM failure
2 fast blinks / 2 second pause	Flash Boot Block Checksum failure
3 fast blinks / 2 second pause	Firmware Checksum failure
4 fast blinks / 2 second pause	MAC address error (error reading MAC chip)
5 fast blinks / 2 second pause	PHY access error
6 fast blinks / 2 second pause	Incompatible firmware

Troubleshooting

If your radio fails to establish contact:

- Make sure the antenna is securely attached.
- Change your location or the location of the antenna by a few feet and transmit again.
- Make sure the PC4500/4800 is securely inserted in the PC Card slot.
- Make sure the receiving equipment is turned on and operating.
- Make sure the receiving equipment is properly connected to the host computer

Power Requirements

Table A.4 - Power Requirements

Specification	Value
Operational Voltage	5.0 ±0.25 Volts
Receive Mode Current	280 mA
Low Power Transmit Mode Current	400 mA at 50 mW
High Power Transmit Mode Current	490 mA at 100 mW
Standby Mode Current	5 mA

Appendix B – PC4500/4800 PCMCIA CIS Description

(Subject to change without notice)

Table B.1 - CIS Information

CIS TABLE		
Offset (hex)	Contents (hex)	Description
00	01	CISTPL_DEVICE - common memory device tuple
01	03	link to 0x05
02	DC	Device ID for common memory space: special function, wait line support, fastest device speed setting
03	00	Device size for common memory space (512 bytes is minimum allocable)
04	FF	Terminate DEVICE tuple
05	17	CISTPL_DEVICE_A
06	03	link to 0x0A
07	DC	Device ID for attribute memory space: special function, wait line support, fastest device speed setting
08	00	size for attribute memory space (512 bytes is minimum allocable)
09	FF	Terminate DEVICE_A tuple
0A	14	CISTPL_NO_LINK
0B	00	link to 0x0C
0C	15	Version 1 Product Information Tuple
0D	12	link to 0x20
0E	04	Major version number required by PCMCIA spec
0F	01	Minor version number required by PCMCIA spec
10-1E	’A’,’i’,’r’,’o’, ’n’,’e’,’t’,0 ‘P’,’C’,’4’,’5’, ’0’,’0’,0	Null terminated manufacturer
1F	FF	Terminate CISTPL_VERS_1
20	20	CISTPL_MANUFACTURER
21	04	link to 0x26
22	5F	AIRONET_MANUFACTURE_TUPLE_LO
23	01	AIRONET_MANUFACTURE_TUPLE_HI
24	05	AIRONET_PRODUCT_NUMBER_LO
25	00	AIRONET_PRODUCT_NUMBER_HI
26	21	CISTPL_FUNCID
27	02	Link to 0x2A
28	06	CISTPL_FUNCID_NETWORK
29	00	System initialization byte
2A	22	Function Extension Tuple
2B	02	Link to 0x2E
2C	01	LAN Technology
2D	07	Wireless
2E	22	Function Extension Tuple
2F	05	Link to 0x35
30	02	LAN Speed
31,32	80,84	
33,34	1E,00	2 Mbps
35	22	Function Extension Tuple

36	02	Link to 0x39
37	03	LAN Media
38	07	2.4GHz Spread Spectrum
39	1A	CISTPL_CONFIG
3A	05	Link to 0x40
3B	01	Size of fields byte: 2 bytes for configuration registers base address, 1 byte for configuration registers presence mask
3C	05	Index number of last entry in the configuration table
3D,3E	E0,03	Base address of configuration registers in attribute memory space as a byte offset
3F	07	indicates the presence of the Configuration Options Register, the Configuration Status Register, and the Pin Replacement Register
40	1B	CISTPL_CFTABLE_ENTRY
41	0C	Link to 0x4E
42	C5	Configuration entry is: the default one; followed by an interface byte; index 5
43	01	Interface description byte: IO + memory interface
44	1A	Feature selection field
45	09	Power selection field: Nominal voltage, Static current
46	55	nominal voltage = 5 volts
47	66	static current = 600mA
48	01	VPP Power selection field – nominal voltage only
49	55	VPP nominal voltage is 5 volts
4A	46	IO space description byte
4B-4D	30,FF,FF	IRQ description bytes
4E,4F	FF,FF	Terminate the tuple sequence

The attribute memory registers are located at locations 0x3E0 and 0x3E2. Only the Card Configuration (0x3E0) and Option Registers (0x3E2) are supported.

Appendix C - Reflashing the Firmware on the PC4500/4800

To reflash the firmware on the PC4500/4800 adapter, use the FLASH.EXE utility. It will correctly interact with the card to accomplish the reflash. However, if you wish to reflash the card without using the FLASH.EXE utility, the following procedure must be used:

1. Apply VCC=5V, VPP1=5V, VPP2=5V.

2. Delay 200ms.

3. Enable output pins and assert RESET.

4. Deassert RESET.

5. Wait for card ready.

6. Read CIS and write 0x45 into the COR (attribute space 0x3E0).

7. Write a value of 0x7E7E into HIF registers 0x28 and 0x2A.

8. Write a value of 0x0010 into HIF register 0x00 (COMMAND register).

9. Poll HIF register 0x00 until b15 is logic '0'.

10. Using the receive mechanism described below, wait for the card to send a * character (0x2A) through the SW_SUPPORT1 register.

11. Using the transmit mechanism described below, send the 'L<cr>' command string through the SW_SUPPORT0 register. Note, the card echos command string characters, so you must also receive characters through the SW_SUPPORT1 register to prevent the system from being deadlocked.

12. Wait for the card to send an <ESC> character (0x1B) through the SW_SUPPORT1 register.

13. Send a 32K block of data to the card using the AUX port. That is, set AUX_PAGE = 0x0100 and AUX_OFFSET = 0x0000, and write 16,384 words of data to the AUX_DATA register. Then, output a value of 0x8000 to the SW_SUPPORT0 register to indicate that the data is valid and ready to be copied to the flash.

14. Repeat steps 12-13 above 3 more times for a total of 4 blocks written.

15. If at any time an error is detected, a return code will be returned through SW_SUPPORT1.

The return codes are as follows:

0x80	Success
0x81	Cannot identify flash
0x82	Error erasing flash
0x83	Error programming flash

To boot the card, follow the previous steps 1-6. Then write 0x0000 into HIF registers 0x28 and 0x2A, followed by steps 8 and 9 above. At this point the card should be operational.

Communicating with the Loader Program

Transmit:

To transmit a byte to the card, use RAM address 0x0028 which translates to the MAC processor SW_SUPPORT0 register. To transmit a byte, first wait until D15 of the word at 0x0028 is clear. Then write the desired byte to 0x0028 with D15 set to logic '1'. D15 is the busy bit - If D15 is set, the transmit register is full. If D15 is clear, you may write to the transmit register without overwriting anything. You may want to implement a timeout while waiting for D15 to clear.

Receive:

To receive a byte from the card, use RAM address 0x002A which translates to the MAC processor SW_SUPPORT1 register. To receive a byte, first wait until D15 of the word at 0x002A is set to logic '1'. Then read the contents of 0x002A. Mask the value read with 0xFF to obtain the data byte. Then, write a value of 0x0000 to address 0x002A to inform the card that you read the data byte.